TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ARI LAPPALAINEN

VIRTUALIZATION APPLICABILITY TO INDUSTRIAL AUTOMATION

Master of Science Thesis

Examiners: Prof. Hannu Koivisto,
Project Researcher Mikko Salmenperä

Examiners and topic approved by the
Faculty Council of the Faculty of
Engineering Sciences
on 13th January 2016

ABSTRACT

Ari Lappalainen: Virtualization Applicability to Industrial Automation
Tampere University of Technology
Master of Science Thesis, 120 pages, 6 Appendix pages
January 2016
Master's Degree Programme in Automation Technology
Major: Information Systems in Automation
Examiners: Professor Hannu Koivisto, Project Researcher Mikko Salmenperä

Keywords: virtualization, industrial automation, security, applicability, implementation, literature review

Industrial automation systems have strict requirements which differ a lot from the requirements of traditional information technology systems. How security is implemented differs as well between the systems. Many components in industrial automation systems were designed for isolated environments and as such they don't have security features on them. These components are still being used in systems because of the long life time of the equipment in industrial automation systems. Because of these reasons, the technologies used in traditional information technology systems cannot be used in industrial automation systems without proper research and testing.

Traditionally different software components are executed each on their separate hardware. This is to provide isolation for software components. The isolation is needed to avoid the bugs and vulnerabilities in one software component from affecting other software components. Using separate hardware for each software component can be costly and hard to maintain. Virtualization is a technology which allows executing multiple software components on same hardware while simultaneously isolating them from each other. Virtualization also provides some advanced features like fault tolerance. These features could be useful in industrial automation systems as well. The problem is that most of the virtualization technologies and solutions are developed for traditional information technology environments. As such research is needed to see if these technologies and solutions can be used for industrial automation.

In this thesis a literature review is done for the existing virtualization technologies and some virtualization solutions are compared with each other. Some existing literature of using virtualization in industrial automation is also reviewed. Based on the literature reviews and the virtualization solution comparisons, the virtualization applicability to industrial automation is evaluated.

The virtualization is found to be suitable to be used in industrial automation systems. However, proper selection of used hardware and virtualization solutions matters. As part of this thesis, a suggestion is made how a virtualized industrial automation system could be implemented. Virtualization adds new requirements for an industrial automation system and using virtualization affects all the steps in the system's lifecycle model. As such virtualization should be taken into account from the beginning when designing an industrial automation system which uses virtualization.

TIIVISTELMÄ

Ari Lappalainen: Virtualisoinnin soveltuvuus teollisuusautomaatioon
Tampereen teknillinen yliopisto
Diplomityö, 120 sivua, 6 liitesivua
Tammikuu 2016
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Automaation tietotekniikka
Tarkastajat: Professori Hannu Koivisto, Projektitutkija Mikko Salmenperä

Avainsanat: virtualisointi, teollisuusautomaatio, tietoturva, soveltuvuus, toteuttaminen, kirjallisuuskatsaus

Teollisuusautomaatiojärjestelmillä on tiukat vaatimukset, jotka poikkeavat paljon tavallisten tietojärjestelmien vaatimuksista. Tietoturvan toteutustavat myös poikkeavat näiden kahden eri järjestelmän välillä. Monet komponentit teollisuusautomaatiojärjestelmissä on suunniteltu eristettyihin ympäristöihin eikä niissä tästä syystä ole tietoturvaominaisuuksia. Näitä komponentteja on edelleen käytössä järjestelmissä, koska laitteiston elinikä on pitkä teollisuusautomaatiojärjestelmissä. Näiden syiden vuoksi tavallisissa tietojärjestelmissä käytettyjä teknologioita ei voi käyttää teollisuusautomaatiojärjestelmissä ilman kunnollista tutkimusta ja testausta.

Tavallisesti useita ohjelmistoja ajetaan erillisillä laitteistoilla. Tämän tarkoituksena on eristää ohjelmistot toisistaan. Ohjelmistojen eristämisellä vältetään ohjelmistoissa olevien koodivirheiden ja haavoittuvuuksien vaikutukset toisiin ohjelmistoihin. Erillisten laitteiden käyttö jokaista ohjelmistoa varten voi olla kallista ja tällaisen järjestelmän ylläpito voi olla vaikeaa. Virtualisointi on teknologia, joka mahdollistaa useamman ohjelmiston ajamisen samalla laitteistolla samalla kuitenkin eristäen ne toisistaan. Virtualisointi voi tarjota myös joitain kehittyneitä ominaisuuksia kuten vikasietoisuutta. Nämä ominaisuudet voisivat olla hyödyllisiä myös teollisuusautomaatiojärjestelmissä. Ongelmana on kuitenkin, että virtualisointiteknologiat ja -ratkaisut ovat tyypillisesti kehitetty tavallisiin tietojärjestelmiin. Tästä syystä täytyy tutkia, että voidaanko näitä teknologioita ja ratkaisuja käyttää teollisuusautomaatiojärjestelmissä.

Tässä opinnäytetyössä tehdään kirjallisuuskatsaus olemassa oleviin virtualisointiteknologioihin ja joitain virtualisointiratkaisuja verrataan keskenään toisiinsa. Lisäksi käydään läpi olemassa olevaa kirjallisuutta virtualisoinnin käytöstä teollisuusautomaatiojärjestelmissä. Virtualisoinnin soveltuvuutta teollisuusautomaatiojärjestelmiin arvioidaan perustuen kirjallisuuskatsauksiin ja virtualisointiratkaisujen vertailuun.

Virtualisointi arvioidaan soveltuvaksi käytettäväksi teollisuusautomaatiojärjestelmissä. Tämä vaatii kuitenkin oikeiden laitteiden ja virtalisointiratkaisujen käyttöä. Tämän lisäksi tässä työssä tehdään ehdotus siitä, miten virtualisoitu teollisuusautomaatiojärjestelmä tulisi rakentaa. Virtualisointi lisää uusia vaatimuksia teollisuusautomaatiojärjestelmälle ja virtualisoinnin käyttö vaikuttaa järjestelmään sen jokaisessa elinkaaren vaiheessa. Tästä syystä virtualisointi pitäisi huomioida jokaisessa järjestelmän elinkaaren vaiheessa, kun virtualisoitua teollisuusautomaatiojärjestelmää ollaan kehittämässä.

PREFACE

This Master of Science Thesis was written in the Department of Automation Science and Engineering in Tampere University of Technology as a part of the DIGILE's CyberTrust project. Making this thesis has been a valuable learning experience for me allowing me to learn many useful things of different virtualization technologies and machines in general. The knowledge acquired while doing this thesis will be certainly useful for me as virtualization is a technology that is being used more in different places and it forms the basis for the cloud services.

I'd like to thank Professor Hannu Koivisto for offering me a chance to do this thesis. I also would like to thank him, Mikko Salmenperä and Jari Seppälä for offering me ideas, guidance and feedback during making of this thesis.

Tampere, 27.1.2016

Ari Lappalainen

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABI	Application Binary Interface
API	Application Programming Interface
APIC	Advanced Programmable Interrupt Controller
APICv	Advanced Programmable Interrupt Controller virtualization
AVIC	Advanced Virtual Interrupt Controller
CAN	Controller Area Network
CAT	Cache Allocation Technology
CFS	Completely Fair Scheduling
cgroup	Control group
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DAS	Directly Attached Storage
DMA	Direct Memory Access
DMZ	Demilitarized Zone
DoS	Denial-of-Service
EOI	End Of Interrupt
ERP	Enterprise Resource Planning
ETSI	European Telecommunications Standards Institute
EVB	Edge Virtual Bridging
FCoE	Fibre Channel over Ethernet
FIFO	First In First Out
GENEVE	Generic Network Virtualization Encapsulation
GIP	Generic Interrupt Controller
gPA	Guest Physical Address
GRE	Generic Routing Encapsulation
gVA	Guest Virtual Address
HMI	Human Machine Interface
hPA	Host Physical Address
I/O	Input / Output
ID	Identifier
IOMMU	Input / Output Memory Management Unit
IOTLB	Input / Output Translation Lookaside Buffer
IPC	Inter-Process Communication
IPSec	Internet Protocol Security
ISA	Instruction Set Architecture
iSCSI	Internet Small Computer System Interface
ISR	Interrupt Service Routine
IT	Information Technology
KVM	Kernel-based Virtual Machine
LBA	Logical Block Addressing
LLC	Last Level Cache
LOGIIC	Linking the Oil and Gas Industry to Improve Cybersecurity
LVM	Logical Volume Manager
MAC	Media Access Control
MES	Manufacturing Execution System
MMIO	Memory-Mapped Input / Output
MMU	Memory Management Unit

MSI	Message Signaled Interrupt
NAS	Network Attached Storage
NFS	Network File System
NFV	Network Functions Virtualization
NIC	Network Interface Card
NPT	Nested Page Table
NUMA	Non-Uniform Memory Access
NVGRE	Network Virtualization using Generic Routing Encapsulation
OS	Operating System
PCI	Peripheral Component Interconnect
PCI-SIG	Peripheral Component Interconnect Special Interest Group
pCPU	Physical Central Processing Unit
PF	Physical Function
PIC	Programmable Interrupt Controller
PID	Process Identifier
PLC	Programmable Logic Controller
PMIO	Port-Mapped Input / Output
QoS	Quality of Service
RAID	Redundant Array of Independent Disks
RID	Requester Identification
RTDS	Real-Time-Deferrable-Server
RVI	Rapid Virtualization Indexing
SAN	Storage Area Network
SDN	Software-Defined Networking / Network
SMMU	System Memory Management Unit
SNIA	Storage Networking Industry Association
SR-IOV	Single Root Input / Output Virtualization
SSH	Secure Shell
SSL	Secure Sockets Protocol
STT	Stateless Transport Tunneling
TLB	Translation Lookaside Buffer
TLS	Transport Layer Security
TPM	Trusted Platform Module
TXT	Trusted Execution Technology
vCPU	Virtual Central Processing Unit
VDC	Virtual Device Context
vDS	Virtual Distributed Switch
VEB	Virtual Edge Bridge
VEPA	Virtual Edge Port Aggregator
VF	Virtual Function
VFS	Virtual File System
VLAN	Virtual Local Area Network
VM	Virtual Machine
VMCB	Virtual Machine Control Block
VMCS	Virtual Machine Control Structure
VMDq	Virtual Machine Device queue
VMM	Virtual Machine Monitor
vNIC	Virtual Network Interface Card
VPN	Virtual Private Network
VXLAN	Virtual eXtensible Local Area Network

1. INTRODUCTION

Computer systems are a lot more complex nowadays than they used to be. Different pieces of software have different features and the pieces of software communicate with each other to produce useful information and to create useful actions. The problem with the complexity is that it is easy to make errors when coding a complex piece of software. Because of this the software can contain coding errors and security vulnerabilities. These coding errors and vulnerabilities can affect the execution of other pieces of software if the underlying hardware is shared between the pieces of software. To minimize the effects of these coding errors and vulnerabilities, pieces of software are often set to execute on their own dedicated machine in traditional information technology (IT) systems. This can be costly as each piece of hardware requires electricity to function and each piece of hardware has to be bought separately [22][76]. Maintaining such systems can also be complicated [76][94].

To solve these problems a technology called virtualization has been developed. *Virtualization* can be considered as a technology where one creates an emulated version of some already existing entity. This allows separating a resource or a service request from its actual implementation [116]. Virtualization allows sharing same hardware between different pieces of software while simultaneously isolating them from each other. A new trend in traditional IT environments has been to use virtualization to reduce the costs of building and maintaining IT systems. Many different virtualization methods and solutions have been developed over the years to solve different kinds of problems. Common virtualization technologies are network, storage, hardware, operating system –level, and application virtualization technologies.

Industrial automation systems typically differ a lot from the traditional IT systems. The reason for this is the strict requirements for these systems. Industrial automation systems control physical processes and any unpredicted behavior can cause damage to environment or people. Because of this many different hardware and software solutions have been developed that can meet the strict requirements of these systems. The solutions differ a lot from the solutions used in traditional IT environments as these environments don’t have similar requirements. This has caused there to be many proprietary solutions for the industrial automation systems which makes these systems often heterogeneous. Traditional IT environments typically use mass produced general purpose hardware and software instead. Other difference is that the industrial automation systems have typically worked in isolated environments while traditional IT environments are often networked with other systems. As a result many of the solutions developed for the industrial automation systems don’t have any security features.

A new trend in industrial automation is to adapt technologies used in traditional IT environments to the industrial automation systems. Cost savings can be achieved this way. The challenge with adapting these technologies is the different requirements of the systems. A technology is not necessarily suitable for industrial automation systems simply because it cannot fulfil the requirements the industrial automation systems have. One example of adapting a technology from traditional IT systems to industrial automation systems is the Industrial Ethernet.

There is interest to adapt virtualization to industrial automation systems in hope of cost savings and ability to use the features provided by many different virtualization solutions. These features can include fault tolerance methods and methods for backing up all the contents of a machine. However, the virtualization applicability to industrial automation must be first evaluated before the existing virtualization solutions can be used in the industrial automation systems.

The goal of this thesis is to review the virtualization technologies used in traditional IT environments, to analyze if they are suitable for industrial automation, and to suggest how a virtualized industrial automation system could be implemented. To do this the requirements, the architecture, the equipment and the security of traditional industrial automation systems are explained. After this a literature review of existing virtualization technologies is done. The principles how these technologies work and what advantages and disadvantages they have are described. Previous literature of using virtualization for industrial automation is also reviewed. Some of the existing virtualization solutions are compared to see what features they offer that could be useful or are required for virtualization in industrial automation. The literature reviews and the virtualization solution comparisons are used to analyze the virtualization applicability to industrial automation.

The Chapter 2 describes how the traditional industrial automation systems are done. The requirements, the architecture and the equipment used in the systems are described briefly. Also the security risks, issues and the used security solutions to these issues are described in their own subchapters. The Chapters 3, 4, 5, 6 and 7 focus on describing the different virtualization technologies, how they are implemented, and what advantages and disadvantages they have. The Chapter 3 gives an overview of the main technology concepts used in network virtualization and then a concrete example of a network virtualization solution, where these concepts are being used, is presented. The Chapter 4 describes how storage devices are virtualized and what data virtualization is. The Chapter 5 focuses on hardware virtualization. Hardware virtualization is a technology which focuses on virtualizing physical machines like servers. Virtualizing physical hardware is a complex task and as such there are many different technologies and techniques used in hardware virtualization. These technologies and techniques are described in this chapter. The Chapter 6 describes operating system –level virtualization technology. While the hardware virtualization virtualizes the whole hardware, the operating system –level virtualization virtualizes only the operating systems used on the physical

machines. The techniques how the operating systems are virtualized are described in this chapter. The last virtualization technology is the application virtualization and it is described in the Chapter 7. How the application virtualization differs from the operating system –level virtualization is described along with the different technologies and techniques used in application virtualization. The Chapter 8 focuses on the security of these different virtualization technologies. The security benefits, risks, challenges, approaches and recommendations are described in their own subchapters. In the Chapter 9 the previous literature related to using virtualization in industrial automation is reviewed. The possible benefits of using virtualization in industrial automation are described along with the challenges that virtualization technologies have when applying to industrial automation systems. Some virtualization solutions are then compared to see what features they have. After this the virtualization applicability to industrial automation is analyzed based on the literature review done about the virtualization technologies, the literature review of using virtualization in industrial automation, and the virtualization solution comparisons. Lastly a suggestion is made how a virtualized automation system could be implemented. This is done by describing what considerations and requirements the virtualization adds to the different steps in a lifecycle of an industrial automation system. Some future research suggestions are also made. In the Chapter 10 the conclusions for this thesis are made.

2. INDUSTRIAL AUTOMATION

The industrial automation systems monitor and control physical systems. As a result they have much stricter requirements than traditional IT environments which mostly just manipulate data and don't control physical systems. Any failures to meet the requirements of industrial automation systems can cause damage to environment or even loss of human lives. [35][110]

The concept of real-time is important in industrial automation. There are five different real-time types: hard real-time, firm real-time, isochronous real-time, soft real-time and best effort. [100] These are shown in the Figure 1.

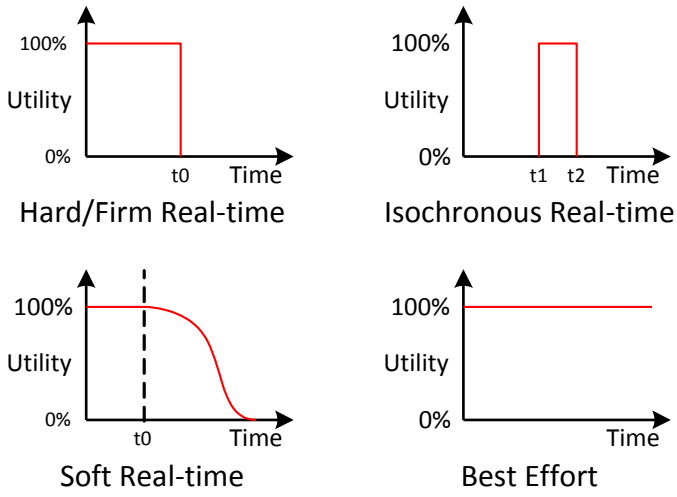

Figure 1. *Different real-time types.*

In hard and firm real-time the usefulness of an action or data becomes zero after a certain time limit. In hard real-time the system fails if the action is not done or the data is not received before the time limit. In firm real-time the system doesn't necessarily fail if the limit is sometimes passed. The action or the data is simply discarded. In isochronous real-time an action or data is useful only within some time period. Outside of this time period the usefulness is zero. In soft real-time the usefulness of an action or data starts decreasing after a certain time limit, but they don't become useless immediately. In best effort there are no real-time limits and actions and data are always useful. Traditional IT environments typically have best effort or soft real-time requirements while some industrial automation systems have hard and isochronous real-time requirements. Control loops typically have isochronous real-time requirements and alarms have hard real-time requirements [100]. In order to meet the real-time requirements, the functionality of the software and the hardware in a system must be deterministic. [2] The determinism can

be indicated with latencies and jitter. *Latency* indicates the average execution time of some action and *jitter* indicates the variance of this value [35]. The actual time limits can vary a lot between different applications [72]. Some industrial automation applications might require cycle times of tens of microseconds with maximum jitter of couple microseconds and some applications might require cycle times of 10 milliseconds with maximum jitter of 100 microseconds.

Other important requirements for industrial automation systems are dependability, reliability, availability and security [12][100]. The industrial automation system should be safe to use and they should not cause any harm to humans or to environment. For this reason safety certified components are used and redundancy [12] is introduced to the systems to mitigate the effects of different kinds of faults. The industrial automation systems are also expected to work continuously for long periods of time as a stop in the production can cause large monetary loses.

A generic industrial automation system model consists of five different levels. Each of these levels has its own functionality. The model is shown in the Figure 2 and it is based on the ISA-95 automation model [98].

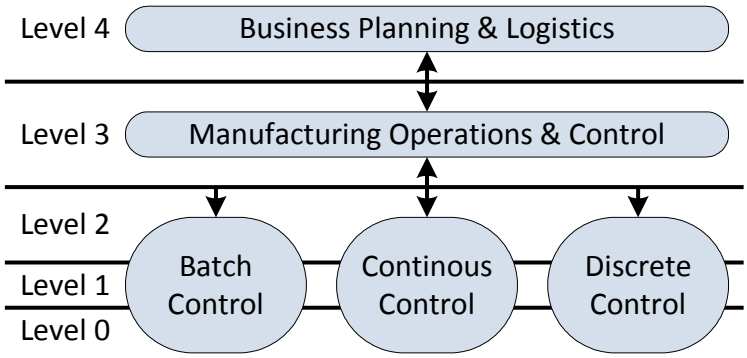

Figure 2. *The structure of a generic industrial automation system.*

The model can be divided into two different parts: the enterprise domain and the control domain. *The enterprise domain* consists of the level 4. This level focuses on managing the whole organization and all the manufacturing plants belonging to the organization. The focus is at business planning and logistics. [98] This level has the least strict requirements and the systems at this level are traditional IT environments [100]. The time frames for operations can be anything from days to years. *Enterprise Resource Planning* (ERP) software components typically work at this level. *The control domain* consists of the rest of the levels. The level 3 is focused on managing the manufacturing process in one manufacturing plant. This contains tasks like product scheduling, monitoring and resource management. *Manufacturing Execution Systems* (MES) typically operate at this level although some of the functionality might be spread to the other levels. The levels 0-2 are focused on the actual process control. [98] The level 0 contains the actual physical processes which are being controlled. The level 1 contains any sensors and

actuators which are used to measure and to manipulate the processes. The controllers which monitor and control the processes are at the level 2. [12] In the control domain the time frames for operations can be anything from microseconds to days depending on the processes which are being controlled [98].

The equipment used in industrial automation systems differ a lot from the equipment which is used in the traditional IT systems. Many of the components in the industrial automation systems must operate in harsh environments with chemicals, dust and physical forces. Ordinary consumer products are not designed for such environments. [35] In addition the strict requirements placed on the automation systems means that specialized equipment must be used which can fulfil those requirements. For example dedicated specialized embedded devices are almost always used at the levels 1 and 2 as the controllers instead of general purpose computers. Other difference between the traditional IT system and the industrial automation systems is that the equipment in industrial automation systems is heterogeneous with many proprietary technologies [100] while the equipment in traditional IT systems is more homogenous and general purpose. Because of this the equipment in industrial automation systems generally has fewer features and is pricier than general purpose equipment. Some equipment that is used in industrial automation systems are programmable logic controllers (PLCs), remote and master terminal units, input/output systems, fieldbuses, sensors and actuators, data management and processing servers, human machine interface (HMI) and intelligent field devices. The fieldbuses use their own proprietary protocols for transferring the messages [26]. The lifecycle of the equipment in industrial automation can also be long and the same equipment can be in use for decades while in traditional IT environment the equipment can change every couple of years. As a result of this there are many old devices which can require old operating systems and machines which are used to configure the old devices [100]. The same equipment is used in industrial automation systems for long time because changing the equipment requires stopping the production and integrating the new equipment to an existing system made out of heterogeneous components can be challenging.

Because the equipment is pricey, there have been attempts to adapt technologies from IT systems to the industrial automation systems. One example of this is the *Industrial Ethernet* technologies [12]. These technologies aim to use Ethernet technology to transfer the real-time data simultaneously with non-real-time data. Because of the requirements of the industrial automation systems many of the Industrial Ethernet solutions still require special technologies which make them pricier [35].

2.1 Security

As the industrial automation systems differ a lot from the traditional IT systems, how the security is implemented also differs between these two systems. This chapter focuses on the security of the industrial automation systems.

2.1.1 Challenges, Risks and Issues

Typically there hasn't been any reason for the automation systems inside the manufacturing plants to communicate with the outside world and the control domains have worked as isolated environments. Because of this many of the equipment developed for the levels 3, 2 and 1 don't have any security features implemented in them. Another reason for the lack of the security features is the fact that they would negatively impact any real-time and timing capabilities of the equipment. [110] The limited processing power in many embedded devices doesn't allow implementing security features [35][110]. Many of the protocols used in the control domain are also transferred in plain text format [110]. Some passwords might be used for the software in the systems, but they are not used for security. They are used to prevent accidental changes and as such the passwords might be hardcoded to the software and there is no way to change them. [100] However, these things have changed with newer equipment. The increased processing power in machines and embedded devices and the increased need for networking with the systems mean that most of the new equipment has some kind of security features implemented in it. However, because of the long life cycle of industrial automation equipment, much of the older equipment without security features is still in use.

PLCs' firmware and operating systems can have bugs or new functionality can be added to them. This requires patches to be applied to the operating systems and firmware. The PLC's don't typically have any kind of validation for the correctness of the software that is applied to them. The patches are validated that they work correctly before releasing them, but if any of the patches is infected with some malicious code at some point, the firmware or the operating system of the PLC could become infected when the patch is applied to the PLC. This can alter the functionality of the PLC. Similarly any control programs which are uploaded to the PLCs don't have this kind of validation either [35]. Some malicious person could upload malicious control programs to the PLCs. The PLCs can also lack any access control to them.

The long lifetime of the equipment in the industrial automation systems also causes risks. As the result of the long lifetime some of the software used can be really old and already considered obsolete outside the industrial automation systems. The software might contain vulnerabilities or bugs which will never be fixed. [110] Other reason why software might have vulnerabilities or bugs is simply because they haven't been updated. The automation systems often run constantly and closing down the system for maintenance is avoided when a new patch is released [35][110]. The maintenance could cause huge monetary losses as the production is halted. In addition patches are often delayed as the system is already working and applying the patch might in some cases alter the functionality of the system. The problem with applying changes to the software is that they often have to applied to the live system because lack of proper testing system [110]. The testing is hard because the industrial automation systems are made out of heterogeneous components with complex interactions and each system is made out of

different components. While some patch might work perfectly fine in some industrial automation system, it could cause problems in another industrial automation system because of different components that have been used.

There are two trends currently ongoing in the industrial automation systems: using Industrial Ethernet to connect the industrial automation systems to other systems and using *Commercial Off-The-Shelf* (COTS) technologies instead of specialized industrial automation technologies. Industrial Ethernet allows more freedom with how the industrial automation systems are connected to other systems. For example the systems can easily be connected to the Internet as the Internet uses Ethernet. Connecting the automation system to the Internet is a huge security risk as the automation systems may lack any security features. Using the COTS technologies in automation systems would allow lowering the costs of building a system as the components used in COTS technologies are mass produced and have a lower price point than components used in specialized industrial automation technologies [100]. The lower price allows using more powerful components which on the other hand allows using more advanced features provided by the software developed for the COTS technologies. However, using COTS technologies in automation systems also exposes them to the security risks the COTS technologies have. This includes viruses, malware, software vulnerabilities, and so on.

People can cause risks as well. The automation systems are generally operated by plant technicians and process engineers. This means that they are not IT security experts. [110] Someone might unintentionally or intentionally spread virus or malware into the automation system with a memory stick or with something similar.

2.1.2 Security Approaches and Recommendations

A common way of providing security to the industrial automation systems is to isolate the part of the system, which doesn't have security features, from rest of the system. This is done by dividing the system into zones and by adding *a DeMilitarized Zone* (DMZ) in between different zones as a buffer. The DMZ should not allow any traffic to go through it. The traffic can only go into it or out of it. Unsecure and secure networks are connected to the DMZ by adding firewalls between the networks and the DMZ. This means that the unsecure and the secure networks can communicate with the DMZ and the DMZ can communicate with them, but the unsecure and the secure networks cannot directly communicate together. If an unsecure network needs any data from a secure network, the data is first transferred to a machine at the DMZ and then the unsecure network can request the data from this machine. [35][110] The concept of the DMZ is shown in the Figure 3 on the next page.

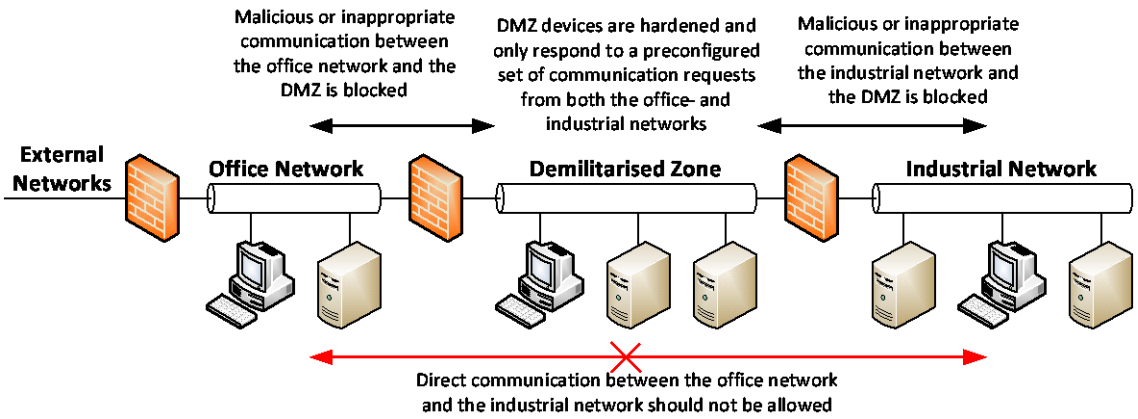

Figure 3. *An example of DMZ implementation. Modified from [35].*

In industrial automation the DMZ is often placed between the levels 3 and 4, and called as a level 3.5 [74]. However, the DMZ can also be implemented in other places as well. For example in between the external networks and the office network could be another DMZ. The traffic is strictly controlled and monitored between the DMZ and the networks connected to it. Security tools can be applied to the edges of the DMZ. For example intrusion detection systems and antivirus tools can be implemented. Additionally *virtual private network* (VPN) endpoints can be implemented to allow secure connections to be created over the networks. Instead of having to implement the security at every device connected to a network, the security can just be implemented at the edges of the DMZ. This allows the security to be managed more easily. [35] At the same time any problems that implementing security features to the real-time applications can cause (losing determinism and slowing down execution) can be avoided. The machines in the secure networks of an industrial automation system are not used for general purpose computing [135]. This lowers the risk of the machines getting infected by malware or viruses which spread through applications like email. The machines in the secure network should be managed and installing any unnecessary software into them should be prevented. In practice a pure DMZ, where connections between the networks connected to the DMZ are not allowed, is rarely implemented. Often VPN connections are made which allow connecting to the industrial network directly from some other network without the need to use the DMZ.

Industrial automation systems are relatively static and deterministic with only small changes to them during their operation. This can be used as an advantage when designing the security for the industrial automation systems. Any information known about the operation of the industrial automation systems can be used when designing the security for them. [135] For example the message types and formats that are transferred in the automation networks are known in advance. This can be used to filter out any messages which don't use these message types and formats. The networks can also be configured in advance without relying to dynamic configuration algorithms. Any dynamic behavior generally lowers the amount of determinism the system has.

Hardening, non-networked safety and fault-tolerance mechanisms, and redundancy are also used to implement security in industrial automation systems. In *hardening* any unnecessary features, services or hardware are removed or disabled from the system. This limits the attack surface the system provides as there are less features, services and hardware in use which can contain vulnerabilities [35]. Non-networked safety and fault-tolerance mechanisms are implemented outside the network which is used for the data communications [135]. As these mechanisms are *out-of-band*, they won't be affected by attacks like *Denial-of-Service* (DoS) attacks. Redundancy is implemented by having multiple pieces of hardware and software which implement the same functionality [12]. With redundancy there is backup software or hardware to rely on if some software becomes infected or hardware becomes faulty. The security is increased by using replicates which don't use exactly the same hardware or software. This reduces the risks caused by bugs in software or manufacturing errors in hardware. The security of the industrial automation systems should be documented properly [110] and the users of the system should be trained about the security issues and policies for the automation systems.

3. NETWORK VIRTUALIZATION

There are many different technologies and protocols which are used to create networks and to connect machines to each other. As such networking is a complex subject which makes network virtualization complex subject as well. Because of this, giving a detailed description of network virtualization is out of the scope of this thesis. In this thesis only the main concepts required to implement network virtualization are presented. There are four important concepts which enable virtualizing physical networks: labelling, quality of service, software defined networking, and device virtualization. These concepts are briefly described in this chapter and lastly a concrete example of where these concepts are used is presented.

3.1 Labelling

In order to separate a physical network into multiple virtual networks, labelling is needed. *The labelling* is used to identify to which virtual network the data transferred in the physical network belongs to. A label is added to packets that are transferred in the physical network. This label can then be used in different network devices to decide where to forward the data next. There are multiple different protocols which can be used to label the data. These protocols include Virtual Local Area Network (VLAN) [50], Q-in-Q [50], Virtual eXtensible Local Area Network (VXLAN) [71], Network Virtualization using Generic Routing Encapsulation (NVGRE) [37], Stateless Transport Tunnelling (STT) [25], and GEneric Network Virtualization Encapsulation (GENEVE) [39]. These labelling protocols can be divided into two categories: tunnelling protocols and non-tunnelling protocols. *The tunnelling protocols* first encapsulate packets by adding a label and some metadata to them. Then they transfer the packets by using the protocols offered by the physical network. The advantages of these protocols are that the protocols used in the virtual network can be different from the protocols used in the physical network and only the edge network devices have to understand the labels. The disadvantage is that these protocols will add overhead to the transferring of data as there is more data that is needed to be transferred over the network. *The non-tunnelling protocols* also add labels and metadata to packets. The difference is that all the physical network devices understand these label protocols natively. This allows less overhead for the data transfers, but the protocols used in the virtual networks must be same as the protocols used in the physical network and all the network devices must understand the label protocol. The difference between the tunnelling and non-tunnelling protocols is shown in the Figure 4 on the next page. VLAN and Q-in-Q are non-tunnelling protocols and the VXLAN, NVGRE, STT and GENEVE are tunnelling protocols.

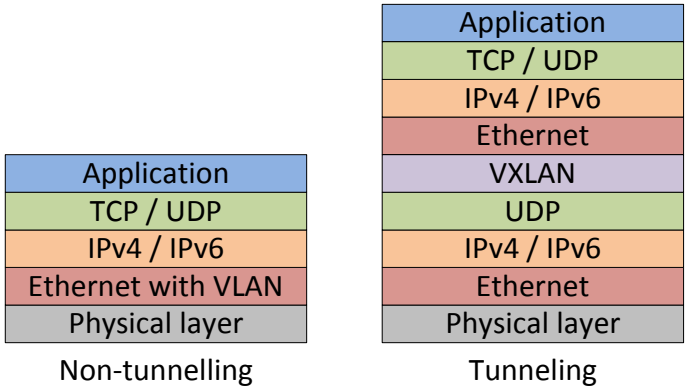

Figure 4. *An example of tunnelling and non-tunnelling protocols.*

3.2 Quality of Service

The labelling allows creating virtual networks by identifying to which virtual network the data transferred in the physical network belongs to. In order to share the physical resources and in order to isolate the performance between the virtual networks, *Quality of Service* (QoS) functions must be implemented. QoS functions are used to make sure that each virtual network has at least certain amount of bandwidth to use in the physical network.

Typical QoS services are based on queues and each port on a network device can have multiple queues attached to it. Packets are inserted into certain queues based on the information embedded in the packet headers and then a scheduler selects a packet from one of the queues to send through the physical medium attached to a port. The scheduler can use many different kinds of algorithms to select the next packet to be sent [89]. The scheduling is important since each virtual network should be able to get to use the medium at some point of time. Additionally the queues can have bandwidth limits [20]. One example how bandwidth limits can be implemented is a technique called *token bucket*. A token is inserted into the token bucket in certain time intervals. These tokens represent a right to send a certain amount of bits through the medium. If the token bucket becomes full, the extra tokens are discarded. On the other hand if the token bucket becomes empty, the packets which arrive to the queue need to either wait for tokens to become available or they can be discarded. [17] This allows small bursts of data be sent thanks to the token bucket's ability to collect tokens while still limiting the amount of bits which one queue can send through the medium. The QoS should be supported on virtual and physical network devices on the whole network.

3.3 Device Virtualization

Another important concept is the device virtualization. The device virtualization can easily be done by modifying any software that resides inside a network device to work on general purpose machine. When this has been done, the modified software can be

virtualized by using some other virtualization technologies like hardware virtualization or operating system –level virtualization. It is also possible that the other virtualization technologies are not needed and multiple virtual network devices can be created simply by launching multiple copies of the network device software.

One example of network device virtualization is *virtual switches*. These are switches that work on general purpose machines and which enable switching between *virtual machines* (VMs) on those machines. The 802.1Q standard specifies standard ways how this kind of virtual switches can be implemented for Ethernet [49]. The actual implementation of the virtual switches varies based on the technologies that are used in the networks. The Appendix A describes implementation of virtual switches. *Virtual Device Context* (VDC) is an example of a technology where parts of the software of a network device are duplicated inside the network device itself to create multiple logical network devices [114]. There is also *Network Functions Virtualization* (NFV) specification which describes how the functions of network devices can be virtualized. The specification describes architecture where functions from network devices are made to work on general purpose machines and which are then virtualized using some other virtualization technology like hardware virtualization to create multiple copies of the network functions. The specification is made by *European Telecommunications Standards Institute* (ETSI). [82]

3.4 Software Defined Networking

Software Defined Networking (SDN) is the last key concept for implementing network virtualization. Every network device has some kind of hardware and software components. The functionality of these components can be divided into three different planes of operation: data plane, control plane and management plane. *The data plane* is responsible for switching packets in the network devices. This is done based on information that is on the headers of the packets the network devices receive. The packets are forwarded to correct destination based on this information. Also some QoS functions are on this plane. *The control plane* is responsible for routing the packets. It decides how the data plane should forward the packets based on information that the control plane collects from the network. Traditional network devices can have distributed algorithms on this plane which collect information about the network to base the routing decision on. Lastly *the management plane* provides functionality that can be used to configure the control plane and to manage the network devices. [32]

In traditional network devices all of these planes reside on a network device itself. This means that each device has its own data, control and management planes. In SDN the control plane and the management plane are moved out from the network devices into some centralized location [76][57]. This allows the routing decisions and the management of the network devices to be made in a centralized location removing the need for distributed algorithms. At the same time the implementation of the network devices

becomes simpler [76][57][43]. Because the network devices in SDN have only the data plane, they are called as switches. The difference between traditional and SDN-capable network devices is shown in the Figure 5. In the figure routers are used on the traditional network, but they could be replaced with any other traditional network devices. It is also worth to note that the SDN controller doesn't have to have the management plane [121]. The management plane could easily be separated to on its own separate management component.

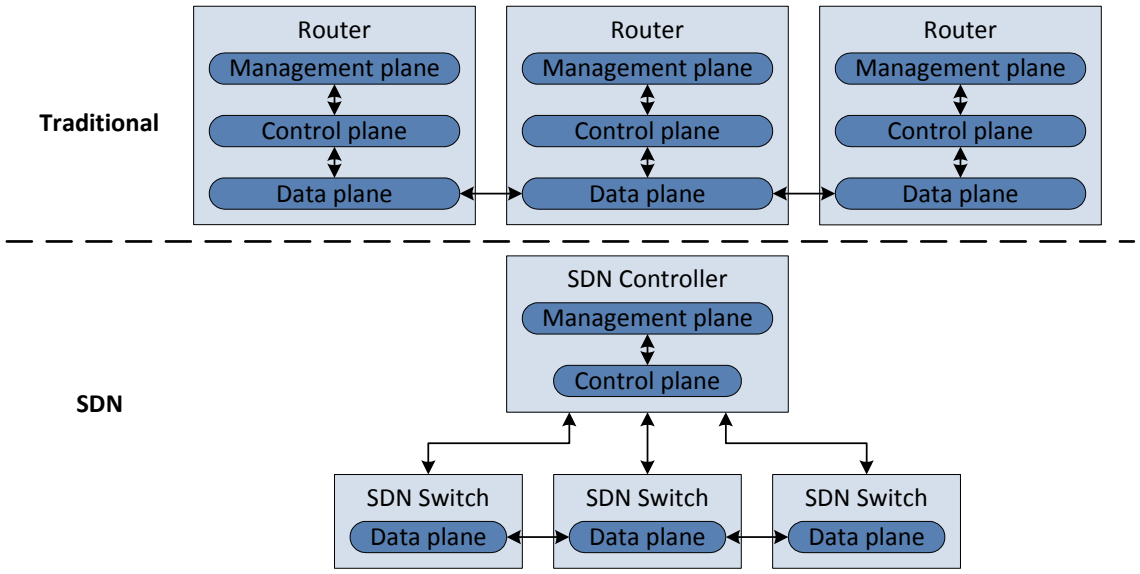

Figure 5. *Difference between SDN and traditional network devices.*

The centralization of the control and the management planes allows creating a logical view of the network structure that differs from the actual physical or virtual structure of the network. As the control plane can collect any data it wants from the network devices, it can easily identify the state of the whole network [76]. For example a software component in control plane or in management plane could combine multiple virtual switches into a one logical switch [121]. This switch could be then presented for the administrator of the network who doesn't have to know the actual structure of the network. The controller makes sure that all the configurations are correct on the actual physical or virtual devices.

3.5 Use Case

A concrete example of network virtualization is the VMware's NSX network virtualization solution [121] though other network virtualization solutions exist. This technology uses all the network virtualization concepts presented in this chapter. The NSX is used on the VMware's hardware virtualization solution to provide network virtualization. The NSX solution is shown in the Figure 6 on the next page. There are two physical machines which both host multiple VMs that are created with the virtualization software.

In the NSX solution VXLAN is used as a labelling protocol to identify different virtual networks from each other [121]. In the figure the VXLAN traffic between the two physical machines is indicated with black arrows. The VMs belong to different virtual networks. The VMs 2, 3 and 4 belong to virtual network 1 and the VMs 4 and 5 belong to virtual network 2. The VM 1 and the NSX controllers belong to special management network. The traffic in these networks is indicated with red, green and purple arrows in the figure. To which virtual network the VMs belong to is based on the ports to which they are connected to in the virtual switches.

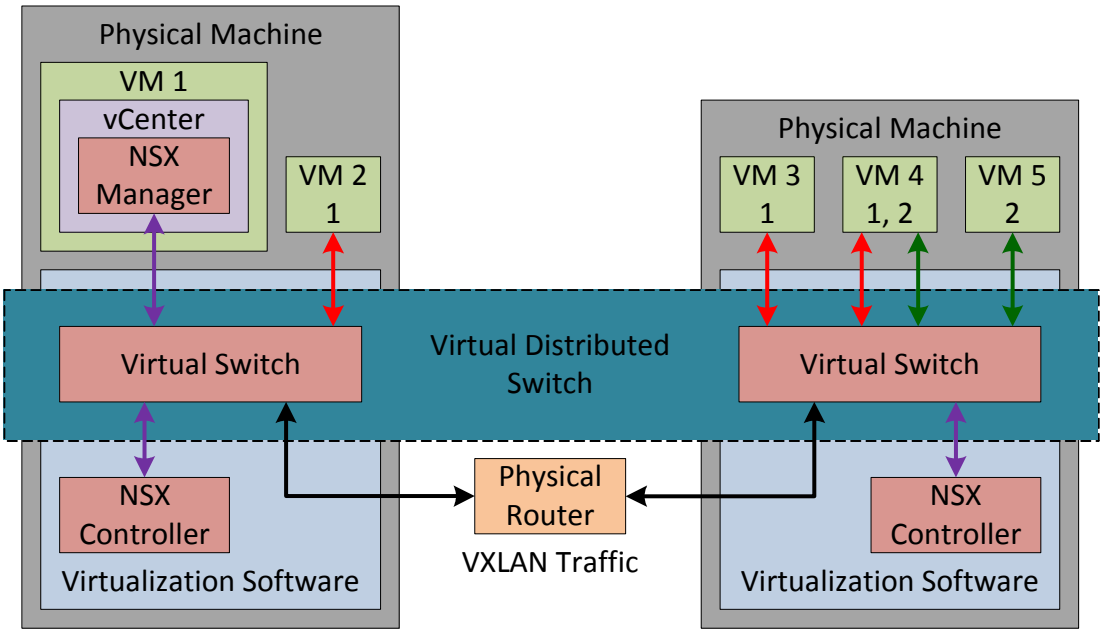

Figure 6. *Network virtualization with NSX.*

Although not shown in the figure, the NSX also supports QoS functions for the virtual switches provided by the NSX. Additionally other virtual network devices like virtual firewalls are also available in NSX to be deployed. The SDN is also implemented in the NSX. There are three important components required for the SDN in the NSX solution. The first component is the virtual switch which supports SDN. The second component is the NSX controller which manages the virtual switches and decides the forwarding rules for the switches. For fault tolerance purpose it is possible to have multiple controllers forming a controller cluster. However, each switch can be controlled only by one controller at a time. The third and the last component is the NSX manager. Administrators can manage the networks through this manager. The manager is a plugin for vCenter management software which is installed on one of the VMs. The NSX combines the separated virtual switches on the physical machines into a one logical switch which is called as *virtual Distributed Switch* (vDS). [121] From a management point of view it seems like all the VMs are connected to a same switch where the traffic is forwarded based on the ports to where the VMs are connected to. Software in NSX makes sure that the traffic is forwarded correctly even if a VM is moved from one physical machine to another.

4. STORAGE VIRTUALIZATION

Storage devices are used to store data for long periods of time. The data is stored as fixed sized blocks containing binary data [76]. These blocks can be accessed by some address which depends on the implementation of the storage device. One example of storage accessing method is the Cylinder-Head-Sector method where the location of a block on the storage device is addressed with a physical reading head of a disk, a cylinder on that disk and a sector on that cylinder [14]. *The Storage Networking Industry Association* (SNIA) has made a model [14] which describes common storage architectures. For the purpose of this thesis a modified version of this model is show in the Figure 7.

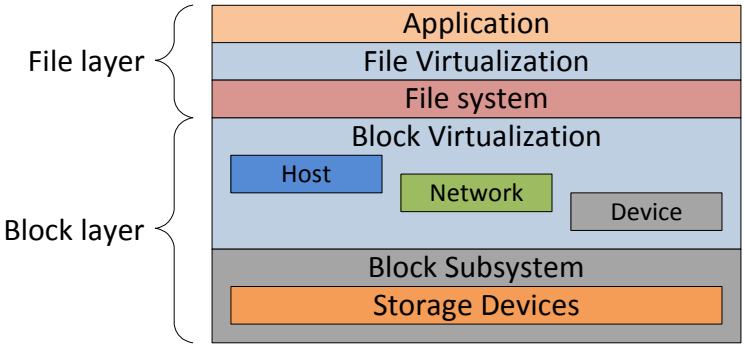

Figure 7. *A model of common storage architecture. Modified from [14].*

Two layers can be identified from the model: a file layer and a storage layer. The block layer handles the blocks the data is stored in on storage devices. The block layer consists of block virtualization layer and block subsystems. The block systems are physical devices which are used to store the data. A block subsystem can contain multiple actual storage devices or it can be just a single storage device. The block virtualization layer is used to create emulated versions of the addresses used to access the storage devices. This virtualization layer can be implemented either on the host machine which uses the storage device, in the network which is used to connect to the storage device, or on the block subsystem itself [14].

The file layer handles files instead of the blocks that the storage devices use. The three components on this layer are file systems, file virtualization layer, and applications. The file systems are the systems which control how the data is stored in the storage devices. In most cases files can be considered as tables which contain addresses to the blocks on storage devices. In addition to these addresses the tables contain also some metadata like a filename and file size. There can also be directories which are similar to the files

[102]. However, unlike the files the directories don't have addresses to data blocks on the storage devices. Instead they have addresses to other directories and files. This allows having hierarchical address space which can be used to access the data belonging to a file. This address space is called as *namespace* as it uses file and directory names to access the data. The namespace hierarchy starts from some *root directory*. [127] There are many different kinds of file systems which can be used on different kinds of storage devices [101]. An example of a file system is *a Network File System* (NFS) which can be used to access storage devices over network by using the namespace addressing method. In some cases the file systems can be directly implemented in storage devices or they can be implemented in the host machine that uses the storage devices. The file virtualization layer is used to create emulated versions of the namespaces the file systems provide. Lastly the applications are just some pieces of software which use the file systems to write and read data in storage devices.

There are few different terms which are common when talking about the connectivity of storage devices. They are *Direct Attached Storage* (DAS), *Network Attached Storage* (NAS) and S*torage Area Network* (SAN). Directly attached storages are just storage devices which are directly attached to the host machine. These storages typically provide block-based interface for the host to use. Network attached storages are storage devices which are used over a network connection. These storages are accessed by using namespaces. Storage area networks are networks which are specifically developed to carry block data for storage devices. This means that the storage devices attached to these networks provide block based interface. [76] The SANs use typically fibre channels to transfer the data. However, there are some technologies which allow transferring the data over Ethernet. Examples of these technologies are *the Fibre Channel over Ethernet* (FCoE) and *the internet Small Computer System Interface* (iSCSI). This means that some of the network virtualization technologies described in the previous chapter can be used in part of the SANs. [106]

4.1 Data Virtualization

Data virtualization is a term used to describe any methods which allow to manipulate and to use data without knowing the actual implementation and the location of the data. [16] There are few important aspects to the data virtualization: unification, abstraction and encapsulation. Unification of the data means that the users don't need to know where the data is located. The data can reside in multiple different storage devices and the user can use the data as if the data resides in one single storage device. The abstraction of data means that the users of the data see only the data what they need. Any data that is not useful for the user is abstracted away. The encapsulation of the data means that the technical details for accessing and using the data are hidden from the user. [65]

The data virtualization can be considered to consist of three different steps:

1. “Connect and virtualize data sources into an abstract format.” [16]
2. “Combine and federate sources into virtual data views.” [16]
3. “Publish these views as a data service for applications or web-based tools” [16]

The implementation of the data virtualization technology depends largely of the data sources and the purposes for which the data is used. However, the model of common storage architecture shows many features required for data virtualization. For example the file systems encapsulate the block access from applications by implementing a new method where files are used to store data instead of blocks. Applications don’t need to know how the actual storage devices work. In addition block virtualization and file virtualization layers have technologies which provide some aspects of unification, encapsulation and abstraction.

4.2 Block Virtualization

Different storage devices have different kinds of properties and because of this the accessing methods can vary. A storage device can virtualize the access method by providing *Logical Block Addressing* (LBA) which can be used to access the blocks on the storage device. This allows one to use indexes to access the data on the storage device instead of using methods like cylinder-head-sector to access the data. [14] Another virtualization method is called as partitioning [101]. In partitioning the storage device is divided into multiple partitions. Each of these partitions functions as if it is its own storage device and has its own file system.

On host machines it is possible to implement software component called *Logical Volume Manager* (LVM). A LVM can be used to implement virtualization for storage devices on host machines [106]. The LVM uses three different concepts to manage the storage devices: physical volumes, volume groups and logical volumes. The physical volumes are actual storage devices or the partitions that are defined in the storage devices. The LVM can group the physical volumes into volume groups each containing separate storage devices and partitions. These volume groups act as if they are physical storage devices. This means that a volume group will have its own address space and the LVM will do address translation to direct the data access to correct partition or storage device. This allows having a virtual storage device which spans over multiple physical storage devices. The logical volumes can be considered as partitions for the volume groups. With the logical volumes the virtual storage devices can be divided into multiple virtual storage devices. Software using the LVM doesn’t need to worry where the actual data is stored and the software can use the virtual storage devices as they would use physical storage devices. [102] The LVM provides unification with the volume groups and abstraction with the logical volumes. It doesn’t provide encapsulation as the

access method to the virtual storage devices is the same as with physical storage devices. The concept of LVM is shown in the Figure 8.

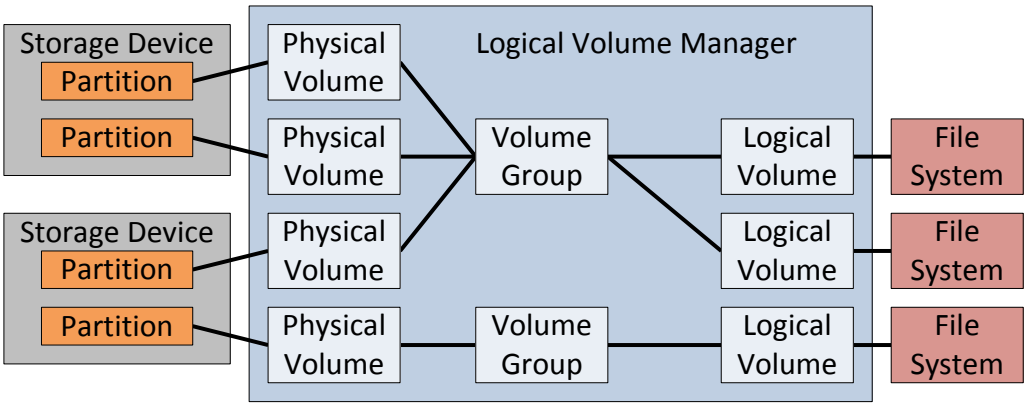

Figure 8. *The structure of a logical volume manager.*

The LVM allows some new concepts to be applied to the storage devices. As one volume group can be made out of multiple physical volumes, it is possible to write to and to read from multiple physical storage devices simultaneously. When data is being written to a logical volume group, the written data can be divided between multiple physical volumes allowing faster writing operations. Similarly the read operations will be faster. This is called as *data striping*. [76] Alternatively the same data could be written to multiple physical volumes simultaneously. This is called as *data mirroring* [106] and it provides fault tolerance as the data is stored in multiple locations. The LVM is cheap to implement as it's just a piece of software and it's also flexible because of this. The disadvantages are that it takes up resources from the rest of the software running on same machine and it is only applicable to the machine it's running on. [76]

The LVM kind of functionality can be also implemented in storage devices themselves. These storage devices have multiple storage devices which are all independent from each other but there is a central controller which controls all of the storage devices [76]. This technology is also called as *Redundant Array of Independent Disks* (RAID) [106]. The RAID storage can be used over a network or they can be locally connected [14]. The advantages of RAID storages compared to the LVMs are that the RAID storages don't take up resources from the software on machines and that the RAID storages can be used by multiple machines. The disadvantages are higher price and the fact that if the centralized components of the RAID storage become faulty, the whole RAID storage becomes faulty [14]. The RAID still provides fault tolerance between the independent storage devices inside the RAID storage.

Block virtualization can also be included in the storage area networks. In SAN multiple storages are connected to a network along with the machines which use them. The storage devices are accessed by using proper addresses which specify where the data is stored. Switches are used to connect the devices together in the network. To implement

storage virtualization in SAN, it is possible to implement special components which implement address translation from logical addresses to physical addresses. These components can either be in-band or out-of-band [14]. Both of these are shown in the Figure 9. The addresses used on the figure don't represent any actual addressing schemes.

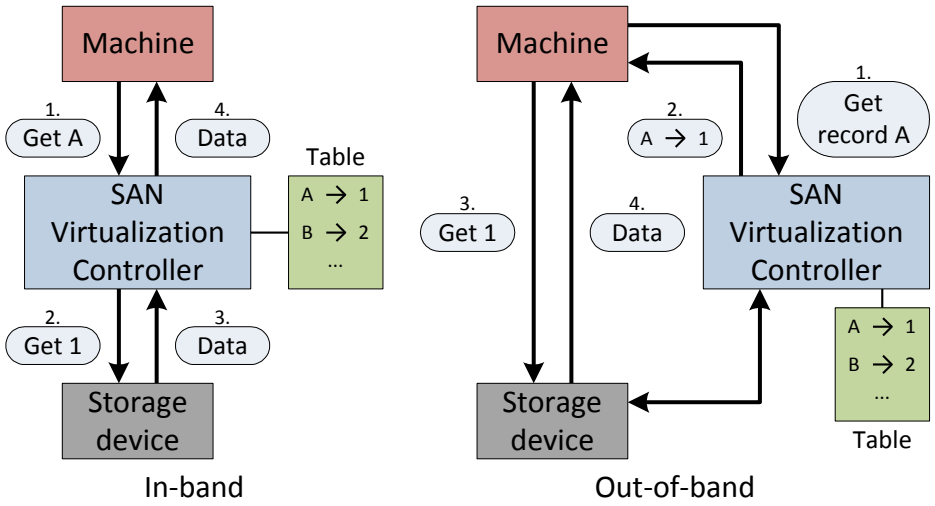

Figure 9. *SAN virtualization. Modified from [106].*

The in-band virtualization component is on the data path [106]. All the data is transferred through it. Each machine sees the virtualization component as its own storage device and the storage devices see the virtualization component as a machine [14]. The component does the address translation and forwards the writes and reads to correct locations. The component allows the machines to see only the storages that belong to them [14] and it can offer additional functionality such as data mirroring or striping. Data caching is also possible as all the data has to go through the virtualization component [76]. The in-band component doesn't require any additional software to be installed on the machines or the storage devices. However, if the virtualization component becomes faulty and fault tolerance is required, a small software component is required on the machines to notice the faulty virtualization component and to redirect the reads and writes to another virtualization component. [14]

The out-of-band virtualization component doesn't reside on the data path. Instead it exists as a separate component from which the storage addresses need to be asked from. This requires a small software component to be installed on machines which makes them aware that the SAN is being virtualized. [14] When a machine wants to use storage device, it has to first ask a proper address from the virtualization component. After this it can use the retrieved address to access the proper storage device. [106] It is possible to have an intelligent switch which retrieves the addresses from the virtualization component as it forwards the data. This way the required software component can be transferred from the machines to network devices. As the data is not required to flow through the virtualization component, the component can be made simpler than the in-band virtualization component [14]. Having an out-of-band virtualization component

allows greater flexibility with data routing than having an in-band virtualization component. Reason for this is that the data doesn't need to go through a single virtualization component. The disadvantage is that functionality like data caching is not possible with an out-of-band virtualization component. Both in- and out-of-band components add some latency from the extra operations they do in networks.

4.3 File Virtualization

While block virtualization technologies handle the blocks the storage devices use, the file virtualization technologies handle the files the file systems use. *A Virtual File System* (VFS) is a file system that is built on top of more concrete file systems. Each volume or partition has its own file system and some file systems are not linked directly to any storage device (for example NFS). Accessing the files on these file systems depends on the implementation of the file systems. A VFS is used to combine the file systems in a way where all the file systems can be accessed in uniform manner. The VFS provides a namespace which can be used to access any file on any file system which has been connected to the VFS and which the VFS understands. The VFS does proper namespace translations so that the file reads and writes are directed to correct filesystems. Unlike the concrete file systems the VFS doesn't handle blocks at all. [62] The VFS provides unification and encapsulation, but it doesn't provide abstraction.

The namespaces of file systems can be virtualized quite easily. As a namespace forms a hierarchical structure of files and directories and the hierarchy starts from a root directory, an emulated namespace can be created by creating a directory and changing it to be the root directory for the duration when the emulated namespace is required. From a root directory it should be impossible to go backwards in the directory hierarchy as the root folder is the first directory in the hierarchy. This means that any software which uses an emulated namespace can only see the files and directories that are in the directory to which the root directory temporarily was changed to. [127] The namespace virtualization provides abstraction but it doesn't provide encapsulation or unification.

5. HARDWARE VIRTUALIZATION

The oldest and the most researched virtualization technology is hardware virtualization. This technology focuses on managing the hardware and virtualizing the interface between hardware and software. This means creating emulated copies of the *Instruction Set Architecture* (ISA). The ISA is an interface through which the hardware can be used, managed, and controlled. It can be divided into two smaller interfaces: the user ISA and the system ISA. *The user ISA* allows using some of the functions and operations offered by the hardware. However, the hardware cannot be configured or managed through this interface. This can only be done through *the system ISA*. [103]

The principle of the hardware virtualization can be seen from the Figure 10. In the figure on the left side is the case where virtualization is not used and on the right side is the case where virtualization is used. In both cases the operating systems are running applications normally. In hardware virtualization a virtualizing layer is added on top of the hardware and below the operating systems [22]. The virtualization layer provides emulated views of the underlying hardware on top of which the virtualization layer resides. This means that the virtualization layer needs to provide virtualized versions of *Central Processing Units* (CPUs), memory, and Input/Output (I/O) devices that the physical hardware has. The virtualized CPU is what provides the emulated ISA for other software to use.

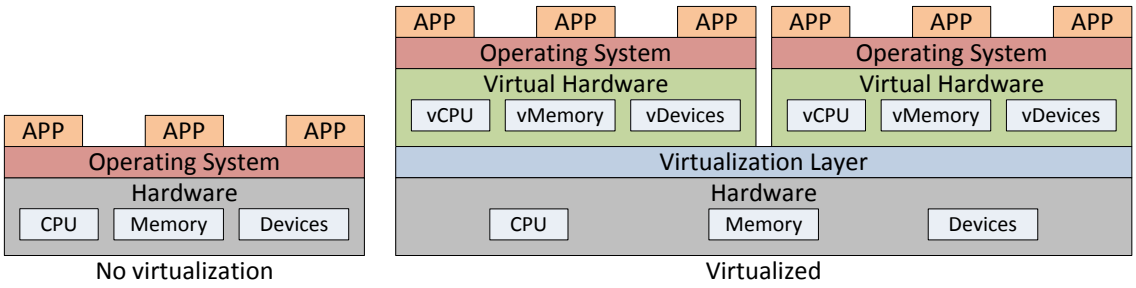

Figure 10. *Hardware virtualization.*

A single set of virtual hardware (CPU, memory and I/O devices) that the virtualization layer creates is called as *a Virtual Machine* (VM) [87]. The software that resides on the virtualization layer and is responsible for providing and managing the virtual hardware is called *a Virtual Machine Monitor* (VMM) [87]. The term *hypervisor* is often used as a synonym for the virtual machine monitor [22][75]. However, VMware uses the VMM and the hypervisor terms to mean different things [116][4]. A VMM is the software component that is responsible for providing and managing the virtual hardware for a single virtual machine. The whole virtualization layer software is called as a hypervisor.

This means that one hypervisor can have multiple VMMs. The hypervisor also contains an operating system. The operating system can either be a general-purpose operating system or an operating system that is specifically developed to run the VMMs. The purpose of the operating system in the hypervisor is to provide a kernel that manages the physical hardware resources for the VMMs. This kernel can also be called as *a vmkernel* if it is specifically developed to run the VMMs [4]. How the functionality is divided inside the hypervisor is not clearly defined. The rest of this thesis uses the VMware's definitions for the terms VMM and hypervisor.

Other useful terms to know are the terms *host* and *guest* [124]. The term host is used to indicate the physical hardware and the software that is not running on virtual hardware while the term guest is used to indicate the virtual hardware and the software that is running on the virtual hardware. For example in the Figure 10 the hardware that is not virtual hardware can be called as *a host machine*, the virtual hardware can be called as *a guest machine*, and the operating systems running on the virtual hardware can be called as *guest operating systems*.

A hypervisor can run multiple VMs at the same time. This means that the hypervisors needs to manage the resources it allocates for the VMs. A VMM is the software responsible for virtualizing the hardware. This sets some responsibilities for the VMM. In 1974 Popek and Goldberg described the properties the VMM should fulfil [87]. To this date these properties can still be considered valid and they are used in modern hypervisors [13]. There are three properties:

1. **The efficiency property**: "All innocuous instructions are executed by the hardware directly, with no intervention at all on the part of the control program." [87]
2. **The resource control property**: "It must be impossible for that arbitrary program to affect the system resources, i.e. memory, available to it; the allocator of the control program is to be invoked upon any attempt." [87]
3. **The equivalence property**: "Any program K executing with a control program resident, with two possible exceptions, performs in a manner indistinguishable from the case when the control program did not exist and K had whatever freedom of access to privileged instructions that the programmer had intended." [87]

The efficiency property simply means that the user ISA instructions should be executed on the hardware directly to allow fast software execution. The resource control property is there to prevent software on VMs from affecting the behaviour of software on other VMs similarly to the way operating systems prevent applications from affecting other applications. The equivalence property means that the software running inside a VM should work the same way it worked without the virtualization. The two exceptions that Popek and Goldberg mention are linked to timing and resource availability problems [87]. In some cases the VMM needs to intervene and modify the instructions the soft-

ware uses from the hardware. This is to make sure that resource control property is kept. The effect of this is that these instructions will take a bit longer to execute than what it would normally take in non-virtualized environment. The resource availability is different in virtualized environment as well since the VMM needs some resources for itself. Additionally when running multiple VMs at the same time, all the VMs require their own resources. Since the software has fewer resources available, it will work slightly differently.

There are multiple ways one can virtualize the hardware. The rest of this chapter focuses on explaining the different virtualization methods and how they are applied to different hardware components. The management of these virtualized hardware components is discussed as well.

5.1 Virtualization Methods

There are three different methods that can be used to virtualize different hardware components [116]. These methods are full virtualization, paravirtualization and hardware-assisted virtualization. All of these methods have some advantages and disadvantages. How the methods are implemented depends on what hardware component is being virtualized.

Full virtualization is a software-based hardware virtualization method [116]. With this method the hardware component is completely virtualized and software can use these virtualized components the same way they would use the components without virtualization. This requires that the hardware components are fully emulated in the VMM which can be quite complicated to implement [13][124]. The advantage of this is that the software used in VMs doesn't need to be modified [116][75]. The disadvantages are that the emulation will affect the performance negatively compared to non-virtualized case and the implementation of VMM becomes complicated.

Paravirtualization is also a software-based method. The difference to full virtualization is that the software running on top of the virtualization layer is modified to be aware of the virtualization layer [116][22]. This breaks the Popek's and Goldberg's third property since the software doesn't function the same way in virtualized environment and non-virtualized environment. However, the benefit is that the performance is greatly improved compare to the full virtualization method [22]. This is because many complicated operations can be replaced with more efficient and simpler virtualization aware operations. The VMM implements a new *Application Programming Interface* (API) called *hypercall interface* [22] to allow the software to use the more efficient operations. The advantages of this method are the better performance and simpler implementation of the VMM than in the full virtualization [116]. The disadvantages are that the software running in the VMs must be modified and different implementations of VMMs can have different hypercall interfaces. Often these modifications mean that the modified soft-

ware can't be used in non-virtualized environment [116]. However, with intelligent design the software can be made to work in both environments [22] and even with different hypervisors as is the case with Linux operating systems that have *paravirt_ops* code in their kernel [129].

Hardware-assisted virtualization is a hardware-based method [116]. The hardware is developed in a way that it can provide emulated versions of itself. The advantages of this method are usually great performance [4][67], the simpler implementation [77] of the VMM, and no modifications required for the software that runs in the VMs. The performance and simpler implementation of the VMM are achieved because most of the functionality can be executed directly on the hardware which is usually faster than executing it in software. The VMMs are still needed to configure the hardware properly for every VM. The obvious disadvantage is that the hardware is required to have the virtualization functionality which is not always available.

When the different virtualization methods are used together, it can be called as *hybrid virtualization* [77][67]. The goal of this is to combine the advantages and disadvantages of the different methods in a way that fulfils the requirements for the system running in the virtual environment. Different VMMs can use different combinations of virtualization methods to provide the virtual hardware for VMs [73][116].

5.2 Hypervisor

The software component in hardware virtualization which manages all the VMs is called a hypervisor. As mentioned earlier, the hypervisors consists of VMMs which provide the emulated hardware for the VMs and a kernel which manages the VMMs.

The hypervisors can be divided into two different types depending on how they are implemented. These types are *type 1* and *type 2* and they both have some advantages and disadvantages. The main difference between these two types is that the type 1 implements everything by itself and the type 2 uses some functionality of an existing operating system kernel. Both of these types are shown in the Figure 11.

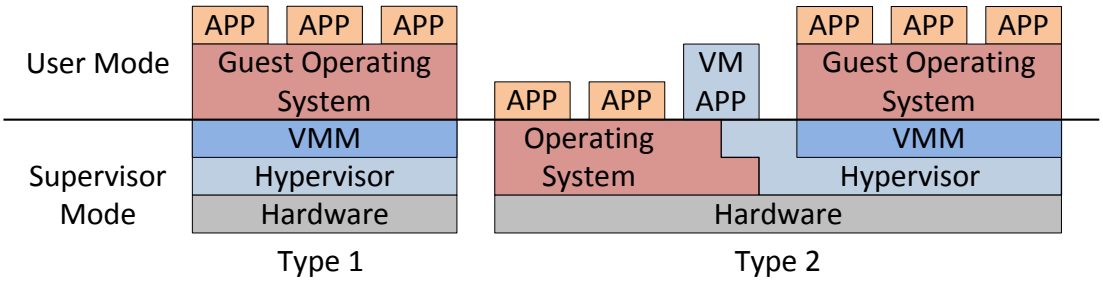

Figure 11. *The type 1 and 2 hypervisors.*

The type 1 hypervisor can also be called as *a bare metal hypervisor* [116] and the type 2 hypervisor can be called as *a hosted hypervisor* [1]. As mentioned, the type 1 hypervisor implements its own kernel to manage the VMMs. This means that the hypervisor controls the usage of the hardware and it can determine how the resources are shared between the VMs. No other software is allowed to use the hardware without permission from the hypervisor to use it. The type 2 hypervisor shares the hardware with an operating system kernel. The sharing is done so that the hypervisor can use the functionality the operating system kernel provides. The operating system kernel is the software which manages the resources and decides which software can use what resources. [1] The hypervisor has to wait for the operating kernel to give it access to use the hardware since the operating system kernel can also host applications other than the hypervisor [13][1]. The hardware resources the hypervisor gets are managed and shared between the VMMs by the hypervisor [1]. The type 2 hypervisor is connected to the operating system kernel with a driver [13] or a kernel module.

The advantage of the type 1 hypervisors is that it is fully in control of the hardware resources. The developer of the hypervisor can determine how all the hardware resources should be managed and shared with the VMMs. Building a special purpose vmkernel also allows limiting the complexity of the hypervisor and the code which is in control of the hardware. [1] Many operating system kernels have functionality which is not required to manage VMMs and are more designed for normal applications. This means that there is code which can manage the hardware, but is not necessarily required for managing and creating VMMs. The hypervisors is also fully dependant of the operating system kernel. If the operating system is required to restart for some reason, the hypervisor must be restarted as well [1]. The disadvantage of the type 1 hypervisor is that it needs to implement everything on its own. This means that native device drivers need to be implemented for each device that the hypervisor is wanted to be able to use [1]. Considering the amount of different devices the computers can have, this is not an easy task. The type 2 hypervisors can simply use the existing native device drivers provided by the operating system kernel they work with [13][1]. It can also run a management application for the hypervisor on the operating system [111] while type 1 hypervisors have to implement their own management tools or provide an API which can be used to manage the VMs. If the API approach is used, a centralized controller can be created which can be used to control hypervisors on different host machines through same management software. Other differences between these two types are that the type 2 hypervisors have worse performance than the type 1 hypervisors [116] and the type 2 hypervisors are easier to install than the type 1 hypervisors. The worse performance of the type 2 hypervisor can be explained by the hardware resource sharing with the other applications as the hypervisor can’t use all the hardware resources for its own functionality and it can’t control the resources by its own [111]. The easier installation of the type 2 hypervisors can be explained with the ability to use the operating system, which is used in

conjunction with the hypervisor to provide VMMs, to install the hypervisor similarly to a normal application [76].

5.3 Central Processing Unit

CPU is the most important component on a computer. It provides the instructions set architecture that can be used to control everything the computer does. It also does the main computing in the computer. To understand how the CPU is virtualized by using the different methods, one needs to be familiar with some basic concepts: privilege levels on the CPU and the types of instructions that the CPU provides.

Privilege levels are used to provide protection for the software using the CPU [44]. Typically CPU has at least two privilege levels: *supervisor mode* and *user mode*. The user ISA is available to use in both modes but the system ISA is only available in the supervisor mode. This means that only the software that is set to run in supervisory mode is allowed to manage the hardware. Arm is an example of a CPU architecture that uses supervisor and user modes [75]. Some CPU architectures like x86 have more privilege levels. In x86 architecture the privilege levels are called as *protection rings* and there are 4 of them numbered from 0 to 3 [116][9]. The ring 0 is the same thing as supervisor mode. Any software running on this ring has unrestricted access to the hardware and to all the instructions. The ring 3 is the same thing as user mode. Any software running on this ring has the most restricted access to the CPU instructions. The rings 1 and 2 are meant for device drivers but they are rarely used [44] and the device drivers are run in the ring 0 instead. Any software running on these rings doesn't have unrestricted access to the hardware and the instructions but it still has more access than software running on the ring 3. It is said that the ring 0 is the most privileged level and ring 3 is the least privileged level [44]. An operating system kernel typically runs at the supervisor mode and user applications at the user mode. The privilege levels can be seen in the Figure 12.

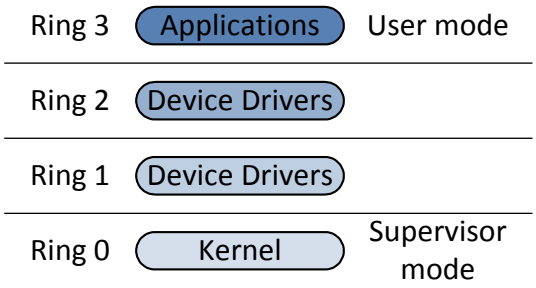

Figure 12. *Privilege levels in different CPU architectures.*

Instructions that the CPU offers can be either privileged or non-privileged and sensitive or non-sensitive. *Privileged instructions* are instructions that can only be executed in the supervisor mode. If a privileged instruction is tried to be executed in any other privilege level, it will cause a general protection exception which will transfer the execution to the software running in supervisor mode [87]. The software in supervisor mode can then

deal with the exception accordingly. This is also called as *instruction trapping*. *Non-privileged instructions* are instructions that can be executed in any execution mode.

There are two kinds of *sensitive instructions*: control sensitive instructions and behaviour sensitive instructions. *A control sensitive instruction* is an instruction which can change the state of the hardware. *A behaviour sensitive instruction* is an instruction which behaviour depends on the state of the hardware or the privilege level of the software that uses the instruction. [87] Rest of the instructions are *Non-sensitive instructions.* Non-sensitive instructions can also be called *innocuous instructions* [87]. The system ISA is made out of the instructions that are privileged or sensitive. Rest of the instructions belong to the user ISA.

5.3.1 Virtualization

As the VMM needs to fulfil the resource control property, it needs to be the only software which has access to the sensitive instructions. This means that there needs to be a method to transfer the control to the VMM when other software tries to use sensitive instructions. Popek and Goldberg created the following theorem:

> "THEOREM 1. For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions." [87]

When all the sensitive instructions are privileged instructions as well, these instructions must be executed in supervisor mode or they will be trapped if they are executed in any other privilege level. Any software using sensitive instructions can then be set to run in any other privilege level other than supervisor mode and a VMM can be created to run at the supervisor mode. This allows the VMM to trap all the sensitive instructions and to guarantee the resource isolation for different VMs. The VMM also needs to emulate the behaviour of the instructions for a VM to make them think they are using the hardware directly and to fulfil the equivalence property. [93] The VMM also stores the state of the CPU which consists of the values of the different registers. This way multiple VMMs can provide virtualization for different VMs. Emulation is not needed for non-sensitive instructions. This kind of virtualization can be called as *classical virtualization* [3] or *trap-and-emulate virtualization* [4]. However, many of the new CPU architectures were not designed with virtualization in mind. These architectures have instructions that are sensitive but aren't privileged. For example both x86 [93] and ARM [24] architectures have these kinds of instructions. Since non-privileged instructions cannot be made to trap, classical virtualization cannot be used. The three earlier mentioned virtualization methods have been developed to solve this problem.

Both full virtualization and paravirtualization move any software running at the supervisor mode to run in some other execution mode. This operation is called as *de-*

privileging [13]. There are two ways the de-privileging can be done: by moving the software to run in some unused privilege level or by moving the software to run in privilege level that is already used by other software. The de-privileging is shown in the Figure 13.

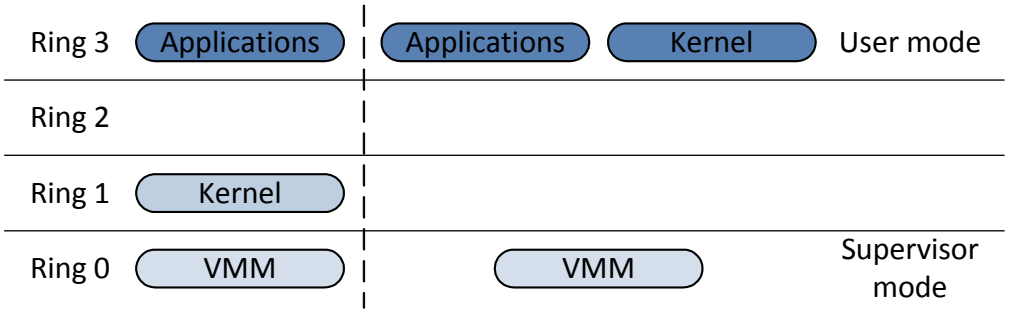

Figure 13. *The different de-privileging methods.*

The method where the software shares the privilege level with some other software has its own challenges. These challenges are discussed more in the Chapter 5.4.1 which talks about the memory virtualization. Because of the challenges this method is rarely used. De-privileging is used to protect the VMM from other software.

Full virtualization typically uses a method called *dynamic binary translation* to deal with the non-privileged sensitive instructions. A binary translator is a piece of software which converts input binary instruction sequence into a second binary instruction sequence that can be executed natively on the hardware. A dynamic binary translator does the same but at runtime. The dynamic binary translator saves the translated instruction sequences into a buffer called translation cache. When the binary translator gets instruction sequence which it has already translated, it can execute the already translated sequence from the cache. This speeds up the translation. In full virtualization the dynamic binary translation is used to identify sequences that have instructions that belong to the system ISA and then translating them into new sequences that can be controlled by the VMM. [13] The binary translation is used for privileged instructions as well since it's faster than trapping [4][73]. These new sequences emulate the behaviour of the input sequences. The instructions that belong to the user ISA are executed directly on the CPU without the intervention of the VMM [13][4]. This allows native execution speed for these instructions. When the translation is done at runtime, the software running on VM doesn't require any modifications. However, the translator needs to check the executed code dynamically causing the performance to be worse than in the native execution of the code. The binary translator can also be quite complicated to implement [13][4].

The way paravirtualization differs from full virtualization is that the instructions which belong to the system ISA are translated when software is compiled instead of doing it at the runtime. The VMM offers a hypercall interface which offers functions that have

similar functionality to the instructions belonging to the system ISA [77][9]. The system ISA instruction calls are then replaced in the software code with these new function calls and the software is compiled. The VMM can manage the hardware this way since any other software uses the functions which the VMM implements. This is much simpler to implement [116] in the VMM than the dynamic binary translator and no extra work is needed during the runtime allowing better performance than full virtualization [116]. The downside is the required modifications to the software that is run on the VMM. Sometimes these modifications can be challenging as the virtual CPU the VMM provides with its hypercall API is different from the physical CPU [77]. Similarly to full virtualization all the instructions that belong to user ISA are executed directly on the CPU without any performance loss.

The implementation of the hardware-assisted virtualization depends completely of the CPU architecture that implements it. However, all the implementations have the same goal: they enable the classical virtualization. Hardware-assisted virtualization doesn't require any modifications to the software that is run in VMs. Here the virtualization extensions for x86 and ARM architectures are explained as an example.

Intel's VT-x and AMD's AMD-V are the virtualization extensions for the x86 architecture processors. They differ slightly with the implementation, but they both have the same basic concept: two new *operating modes* are implemented for the CPU. The operating modes are *root mode* and *non-root mode* [80]. These new operating modes are not an extension to the old privilege levels, but are orthogonal to them. This means that both of these modes have the old privilege levels as can be seen from the Figure 14.

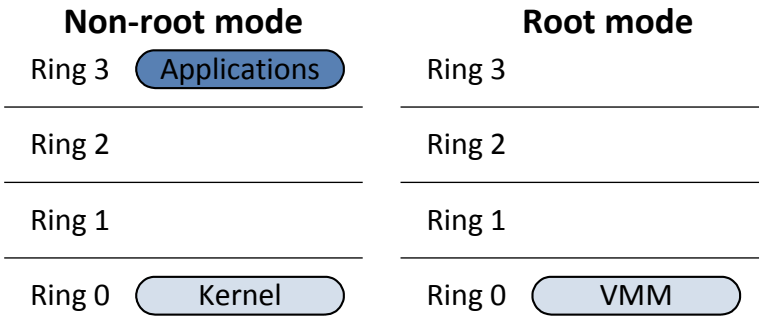

Figure 14. *The new operating modes of the x86 architecture.*

Since the operating modes are orthogonal to the existing privilege levels, de-privileging is not needed for the software that runs in the ring 0 normally. The VMM and the software are set to run in different operating modes so they both can use the ring 0 as shown in the Figure 14. The non-root mode is meant for normal software and the root mode is reserved for the VMM and any software needed for the virtualization. [80]

The virtualization extensions include new in-memory control structures that are used to save the state of the CPU in different operating modes. Each VM has its own control

structure as does the software running in the root mode. In VT-x the control structures are called *Virtual Machine Control Structures* (VMCS) and in AMD-V they are called *Virtual Machine Control Blocks* (VMCB). Every time the operating mode changes, the CPU saves the current CPU state to a control structure and then loads another CPU state from a proper control structure automatically [4][80]. Since the CPU states are saved separately, the software running in ring 0 in non-root mode can use instructions from system ISA normally without the need for intervention from the VMM. This is only true for instructions that only modify or read the state of the CPU. Without hardware assisted virtualization for memory and I/O, the execution must still be transferred to the VMM to virtualize them. [3]

The control structures some contain some other information in addition of the CPU state. They contain fields that can be used to specify what instruction will cause the execution to transfer to the VMM in root mode. They also contain field which is used to store the reason why the execution was transferred to the VMM so the VMM can do right decisions after that. [4][80]

VT-x and AMD-V offer improved performance for instructions that only use the CPU because they can be executed natively. However, changing the execution between the operating modes is slow since the whole CPU state needs to be swapped [4]. This is why operations that transfer the execution to the VMM should be avoided. If the memory and I/O are virtualized in software, instructions that use these require the execution to be transferred to the VMM and the performance can suffer a lot compared to case where the CPU is virtualized in software as well [73].

The virtualization extension for the ARM architecture takes a bit different approach. Instead of having new operating modes it simply implements a new privilege level that is more privileged than the old privilege levels. [118] This new privilege level is called as *hyp mode* [8] and it is show in the Figure 15. The new level is meant for the VMM and the rest of the software can run on its privilege level it was designed for. All the old instructions that are privileged or sensitive can be configured to trap and transfer the execution to the new privilege level or to the old supervisor mode [118]. This allows using the classical virtualization method.

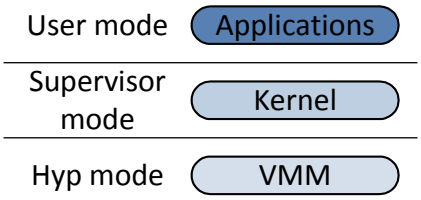

Figure 15. *The new execution mode for the ARM architecture.*

When the execution is transferred into or out of the hyp mode, the VMM needs to save or restore the state of the CPU on its own [118]. The ARM virtualization extension

doesn't do this automatically like the x86 extensions do. To increase the execution speed in virtualization, the virtualization extension includes emulation support. When an instruction is trapped it must be first decoded before its emulation. Normally the decoding and the emulation are done in software. With the emulation support the decoding can be done in the hardware instead which increases the performance [118].

5.3.2 Management

As the hypervisor is responsible for managing the physical hardware, it is responsible for deciding how the CPU resources are used. Each VMM can emulate multiple CPUs. These emulated CPUs are called as *virtual CPUs* (vCPUs). The hypervisor manages how these vCPUs are mapped to the *physical CPUs* (pCPUs). [73] It selects on which pCPU each of the vCPUs is run. To do this the hypervisor must schedule the vCPUs.

The vCPUs can be modelled as threads on multicore systems [4][73]. These threads can be assigned to run on one core or the threads can be set to run on a multiple cores in which case the hypervisor selects suitable cores on which the threads are run. With this it is possible to dedicate a physical core for a vCPU to use. This is also called as *vCPU pinning.* Only the vCPU's thread is set to run on one of the cores and nothing else is set to run on it. This removes the need for scheduling as the vCPU is the only thing that can run on the physical core.

On type 1 hypervisor the scheduler must be implemented by the hypervisor itself. On type 2 hypervisor the scheduler is implemented by the operating system kernel. [73] The operating system kernel must support pinning threads on certain physical cores for the hypervisor to support the vCPU pinning. The scheduler uses some kind of scheduling algorithm to select which vCPU is run next on a physical core. There are many different kinds of algorithms which can be used for the scheduling and explaining all these algorithms won't be part of this thesis. However, it is important to note that with the hypervisor the system becomes a hierarchically scheduled system. The hypervisor uses its own scheduler and the operating system kernels running in VMs use their own schedulers to schedule the guest applications. This is show in the Figure 16 on the next page.

The hierarchical structure allows few options how the scheduling can be managed. The scheduler in the hypervisor can simply schedule the vCPUs without considering how the guest schedulers work or if the guest schedulers have any requirements. Alternatively the scheduler in the hypervisor can take the requirements of the guest schedulers into account and try to schedule the vCPUs in a way that allows the guest schedulers to meet their requirements. Some methods have been developed for this approach. For example compositional scheduling framework can be used to analyse the hierarchical scheduling structure and to determine settings for the scheduler in the hypervisor so that it can meet the requirements for the guest schedulers [131][31]. Another approach is to flatten the

scheduling hierarchy. In this approach the scheduler in the hypervisor uses the knowledge of the tasks running in the guest operating systems to determine which vCPU to schedule next. The hypervisors requires information of the tasks scheduled on the guest schedulers to make its decisions. [31] The information can be provided for example with VM introspection [31] or with paravirtualization.

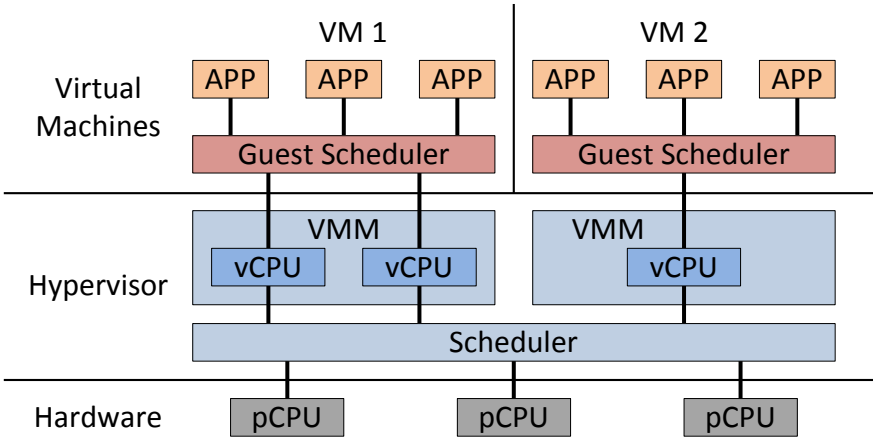

Figure 16. *Hierarchical CPU scheduling.*

Even if the scheduler in a hypervisor doesn't consider the requirements of the guest schedulers, there are some challenges for the scheduling in the hypervisor. These challenges are related to multiprocessor systems. Many operating systems may assume that the processors they use run approximately at the same rate. In virtualization this is not the case as the vCPUs are scheduled on the pCPUs and all the vCPUs belonging to one VM might not be running simultaneously. Operating systems might use spinlocks for their functionality. For example if one of the vCPUs holds a lock and that vCPU is descheduled, the other vCPUs will waste time busy-waiting while waiting for the vCPU holding the lock to be rescheduled and to free the lock. This will make the operating systems work slower than how they would work in non-virtualized environment. The difference between vCPU execution rates is called as *skew*. The solution for this problem is for the scheduler in the hypervisor to schedule the vCPUs to execute at the same time. [73] This is called as *gang scheduling* or *co-scheduling*. These scheduling methods can be relaxed a bit by measuring the skew between the vCPUs and scheduling them accordingly to keep the skew within some bounds. Alternatively the operating systems can be designed to not assume that the CPUs they use run simultaneously.

Another thing the scheduler in the hypervisor should be aware of is the *Non-Uniform Memory Access* (NUMA) architecture. In NUMA architecture the computer is divided into nodes. Each node has a set of CPUs and a memory. The CPUs can access the memory belonging to the same node quickly but accessing the memory in other nodes is much slower. The scheduler in the hypervisors should be *NUMA aware* so it can schedule the vCPUs to run at the optimal pCPUs to provide the best memory access for the vCPUs [73].

5.4 Memory

Memory is used to store data temporarily. It is used because it is much faster to use than some permanent storage solution like a hard disk. Memory virtualization is not a new technology and many operating systems use it [117]. However, when multiple operating systems are run simultaneously on same machine, new methods are needed for the memory virtualization. To understand these methods one needs to know some basic concepts of the memory virtualization technologies the operating systems use. Memory virtualization for multiple operating systems is based on these technologies.

Memory is accessed by using *physical memory addresses*. These addresses point at certain locations in the memory and certain number of bits of data is located at these points in the memory. The addresses grow in successive order and they form *physical address space*. The traditional methods how the operating systems virtualize the memory are segmentation and paging. [52]

In segmentation the memory is divided into fixed length portions. This is done by having two values: *a base* and *a limit*. The base indicates the location from where a segment starts in the physical memory and the limit indicates the amount of physical addresses that belong to this segment. These values are stored in some CPU register. When a segment is created, a new address space is created called as *a logical address space.* This address space is separate from the physical address space. *A logical address* contains two fields: *a segment* and *an offset*. The segment field is used to indicate what segment is used and the offset field is used to indicate which address in that address space is used. When the logical address space is used to access memory, an address translation is needed from the logical address space to the physical address space or, if paging is used, to a linear address space. [52] This is done usually in some hardware component like *a Memory Management Unit* (MMU). The address translation consists of finding the base value that is linked to the segment number from a table and then adding the offset to it [52]. If the offset is larger than the limit, the address translation will generate a fault. A segment can be assigned to a piece of software and the software can then use the offset value to access the software's memory like it would have direct access to the physical memory. The offset can be though as an emulated version of the physical address space. This is why the logical address space can also be called as *a virtual address space*. The details how segmentation is implemented typically vary between different CPU architectures. The concept of segmentation can be seen in the Figure 17 on the next page. The offset validation is not shown in the figure.

The disadvantage of segmentation is that the memory, which is allocated to software, needs to be continuous. In paging the physical address space is divided into multiple equally sized blocks called as *pages*. When paging is done, a new address space is created called as *a linear address space*. *A linear address* contains typically three fields: *a directory, a table* and *an offset*. However, there might be more or less fields in the linear

address depending how many levels the paging has. The directory field is used to indicate an entry in a table called as *a page directory*. This entry has a physical address to a table called as *a page table*. The table field is used to indicate an entry in this page table. This entry has a physical address to a page in the memory. The offset field is used to indicate the physical address inside the page. In cases where there are more levels to the paging, all the tables can be simply called as page tables. The entries in the page tables simply point to other page tables or pages. When the memory is accessed by using a linear address space, an address translation is needed to transform the linear address to a physical address. Page directories and page tables are stored in the memory. The physical memory address to the current software's page directory is stored on a register in the CPU. First the entry from page directory is searched with the value in the directory field and then the page table's memory address, that the entry has, is stored to a register on the CPU. After that the entry from the page table is searched with the value in the table field and then the page's memory address, that the entry has, is stored to a register on the CPU. Lastly the offset is added to this stored memory address to get the physical address. [52] If a page directory, a page table or a page is not found, *a page fault* is generated which transfers the software execution to the software running in supervisor mode so it can decide what to do. The address translation is done usually by hardware component like a MMU. The address translation can also be called as *page walking* [4]. The concept of paging can be seen in the Figure 17.

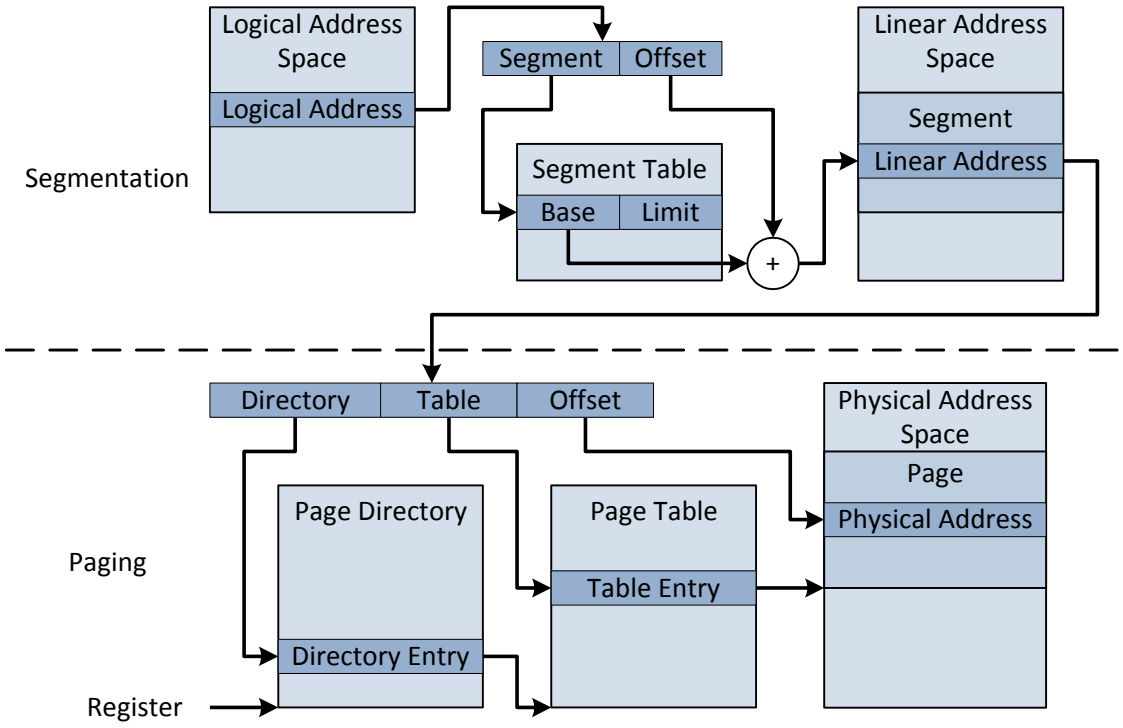

Figure 17. *The typical memory virtualization methods operating systems use.*

Since the address translation needs to access the memory three times to find the correct physical address, it is three times slower than just directly accessing the physical

memory. To solve this problem a special cache called as *a Translation Lookaside Buffer* (TLB) has been developed [52]. It can be implemented either in software or hardware. The TLB is used to store address mappings from the linear addresses to physical addresses. This way the physical addresses can be directly found from the cache and the address translation is not needed thus making accessing the memory faster. When the address translation is done, the translated address is stored in the TLB. However, the size of the TLB is limited and when the TLB becomes full, some stored translated address must be removed to make space for the new address [4]. There are many different algorithms that can be used to determine which translated address should be removed next. When accessing the memory and an already made translation is not found from the TLB, it is called as *a TLB miss.* In this case the memory translation has to be done making the memory access slower.

Each software component can have its own linear address space. This means that the contents of the TLB should be emptied every time the executed software changes. Otherwise the TLB could contain addresses that do not map to the memory that the software owns. The operation where the TLB is emptied is also called as *TLB flushing* [4]. The flushing is often performance heavy operation. However, the flushing isn't needed if there is a way to indicate to which software component the translated addresses belong to. This is done by adding a tag to each translated address and the TLB which implements this is called as *a tagged TLB* [9].

5.4.1 Virtualization

When virtualizing the memory for multiple operating systems, each VM is given its own virtual address space to use. This address space can be called as *guest physical memory* and it's emulated by the VMM. The physical memory is called as *host physical memory.* [117] In the software-based CPU virtualization methods parts of the VMM needs to reside in the guest physical memory [13][80]. The reason for this that the parts of the VMM that transfer the execution from the software running in a VM to the VMM needs to be available to use in this address space [4]. In hardware-based CPU virtualization this is not required as the hardware transfers the execution automatically and software is not needed for that.

Many of the data structures used in traditional memory virtualization contain fields that are used for protection. These are typically linked to the privilege levels of the CPU in one way or another. [52] When the memory is virtualized, there is need for three levels of protection: one for the VMMs, one for the operating system kernels and one for the applications. The VMMs needs to be protected from both the operating system kernels and the applications while the operating system kernels need to be protected from the applications. [48] The protection can be done in one of two ways: by having the software components in separate address spaces or by using the protection fields that the existing memory virtualization methods offer. The CPU architectures that have only

two privilege levels might not have more than two levels of memory protection. If software-based CPU virtualization is used and the operating systems kernels need to be de-privileged, they need to share a privilege level with applications. In these cases the VMM needs to implement the protection between the operating system kernels and the applications by separating them into different linear address spaces [78]. This means that there are two separate address spaces for each VM: one that contains the memory of the whole VM and one that contains the memory of the whole VM apart from the memory that belongs to the operating system kernel [9][88]. By having these two address spaces the operating system kernel can manipulate the whole memory of the VM apart from the memory that belongs to the VMM and the applications can only manipulate the memory that belongs to the applications. In this method when the operating system kernel wants transfer the execution to an application, it needs to do it through the VMM [88]. The VMM will swap the address space to that which doesn't let manipulating the operating system kernel's memory. When the execution is transferred back to the operating system kernel a new swap is required to change the address space back to the other one. If the TLB is not tagged, the address space swapping requires a TLB flush which is performance heavy operation [77]. This is why protection done with protection fields is superior to protection done with address space swapping in systems which don't have a tagged TLB.

As the hardware components that manage the memory are dependant of the CPU architecture they are used with, the memory virtualization methods are also dependant of the used CPU architecture. Some ARM CPU architectures support memory domains [8]. A domain is a collection of memory regions and each memory page can be set to belong to some domain. The access to these domains can be controlled and this way they can be used to protect the software components from each other [48].

In the x86 architecture the segmentation can use the same amount of protection levels as there are protection rings. However, the paging only supports two protection levels: one for the rings 0-2 and one for the ring 3. [52] Also the TLB in the x86 architecture is not tagged. This is why the software-based virtualization methods on the x86 architecture use segmentation to protect the VMMs from the operating system kernels and paging to protect the VMMs and the operating system kernels from the applications [4]. Protecting the VMMs this way can cause some problems. Since a VMM shares the same address space partially with a VM and the segments are controlled by changing the segment limits, the VMM needs to reside at the top most part of the address space. Any software which uses these top parts of the address space cannot use them. [13] This problem is called as *address space compression* [80].

The 64bit extension (x86_64) for the x86 architecture mostly removed the support for the segmentation as nothing really used it after the paging was implemented. This meant that there was no way of protecting the VMMs from the operating system kernels if they were de-privileged to the rings 1 and 2. Later the AMD added the segmentation back to

their CPU chips for the x86_64 extension seeing how it was useful for the virtualization, but the Intel's CPU chips still don't have the segmentation for the x86_64 extension. [4] Because of the lack of the segmentation, the operating system kernels and the applications need to be separated into different address spaces. This is the reason many VMMs don't necessarily support software-based virtualization for the x86_64 extension as it would be slow.

VMMs use the existing memory virtualization methods to virtualize the memory. Since these memory virtualization methods are often implemented in hardware components and many operating systems use them, the VMMs need to emulate these hardware components for the VMs [116]. This is where the virtualization methods listed in the Chapter 5.1 come into play.

In the full virtualization two sets of mappings are required: the operating system kernels map *the guest virtual addresses* (gVAs) to *the guest physical addresses* (gPAs) and the VMMs map the guest physical addresses to *the host physical addresses* (hPAs) [4]. The main technique how emulated paging is implemented in full virtualization is called as shadow paging. In *shadow paging* the VMMs maintain mappings from the guest physical addresses to host physical addresses and mappings from guest virtual addresses to host physical addresses are stored into shadow pages. These pages are then used by the hardware memory management components. [73] When software tries to edit mappings in page directories or page tables in the VM, the execution is transferred to the VMM which validates the edits and creates proper mappings into the shadow pages [104]. Accessing the memory can be done directly since the hardware uses the mappings from guest virtual addresses to the host physical addresses [104]. The concept of shadow paging can be seen in the Figure 18.

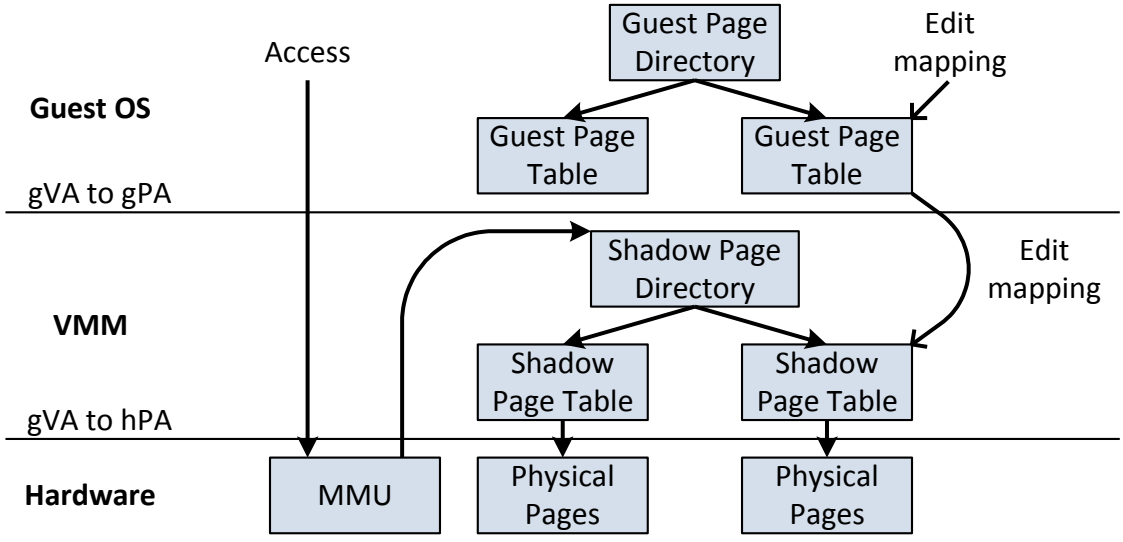

Figure 18. *The shadow paging technique.*

The segment registers, that the x86 architectures uses to store the information about the segments, have some interesting attributes. They have visible and hidden portions. Software running at any privilege level can read or write the visible portion but it can't directly access the hidden portion. When to the visible portion is written, it automatically updates the hidden proportion as well. [13] The hidden proportion contains information like the base address of the segment and the limit. Additionally the x86 architecture has data structures called as segment descriptors. These contain the same information as the hidden portions of the segment registers. [52] If this information is changed in the descriptors, it won't automatically update the hidden portion of the segment registers as the only way to update those is by writing to the visible portion of the segment register. Some operating systems rely on this behaviour so this must be emulated as well. [13]

The vPA to hPA mapping can be applied to segmentation as well. The main full virtualization techniques for emulating the segments in the x86 architecture are shadow descriptors and cached descriptors. *The shadow descriptors* emulate the segment descriptors and *the cached descriptors* emulate the segment registers. They work similarly to the shadow pages. [13]

The full virtualization method for the memory will be slower that executing software directly on the hardware. The reason for this is that the VMM needs to intervene every time some software tries to edit memory mappings [73][104]. Also when the VM accesses memory for the first time, the shadow pages must be created slowing down the memory access as well [104]. Lastly every time the executed VM is changed the shadow pages must be changed to contain the correct mappings for the new VM [104].

In paravirtualization the operating system kernels map the guest virtual addresses directly to the host physical addresses. As the host physical memory space is used by all the VMs, the operating system kernels can't map the addresses freely to the host physical addresses. To emulate the paging the VMMs need to provide hypercalls that can be used to create new pages and to edit the mappings of existing pages. This way the VMMs can validate that the software doesn't try to use memory that doesn't belong to the VM in which the software is running. [9] The VMMs also need to provide a hypercall that can be used to see which physical memory addresses are available so that the operating system kernels can select proper mappings for the pages. The operating system kernels are then modified to use these hypercalls instead of the ISA instructions. To protect the memory, access to the memory through the ISA needs to be set to read-only [9]. Since the operating system kernels use gVA to hPA mappings, they can use the memory management hardware directly to access the memory. The concept of paravirtualized paging can be seen in the Figure 19 on the next page.

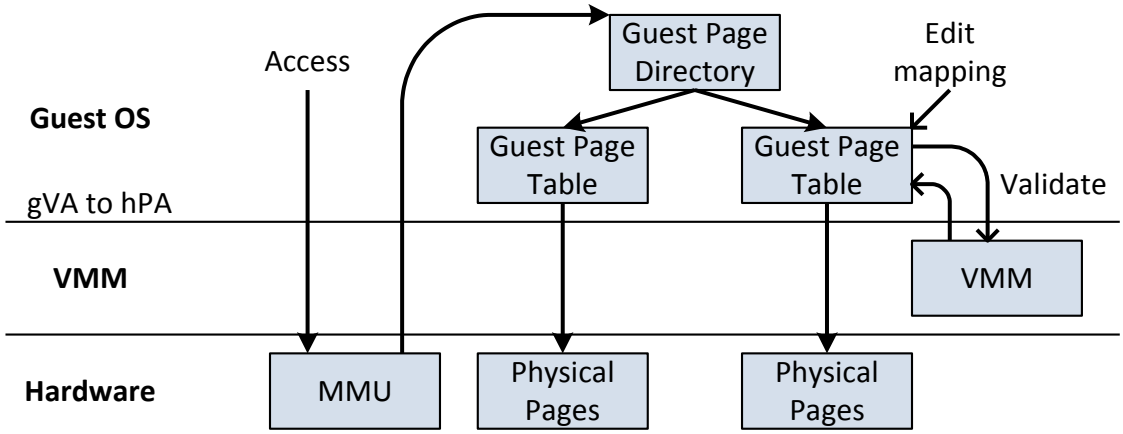

Figure 19. *The paravirtualized paging.*

Segmentation in paravirtualization is virtualized the same way. Any edits to the segment descriptor tables must be done with hypercalls so that the VMMs can validate them [9]. Since the segmentation is used with hypercalls and the software running in a VM must be modified, the problem with relying on the hidden portion of the segment registers not changing can be avoided.

The benefit of the paravirtualization compared to the full virtualization is that the VMMs don't need to maintain shadow pages. This means that only time the VMMs need to intervene and do something is when the guest operating system kernels create new tables or modify address mappings. This allows the paravirtualization to perform better than the full virtualization. At the same time the implementation of the VMM becomes simpler. [9]

In hardware-assisted virtualization the hardware is modified to be able to handle both gVA to gPA and gPA to hPA address translations. For x86 architecture there are two virtualization extensions for memory: Intel's Nested Page Tables (NPT) and AMD's Rapid Virtualization Indexing (RVI) [4][73]. The details of these extensions vary, but the concept is same in both of them. These extensions are part of the Intel's VT-x and AMD's AMD-V extensions, but it is important to note that some of the first CPUs that support VT-x and AMD-V don't have nested or extended page tables. In these extensions the VMMs maintain two sets of page directories and tables: one set maps addresses from vPA to gPA and one set maps addresses from gPA to hPA [73]. When a memory address translation is done, the hardware component first looks up what is the guest physical address of a page table or a page in the entries of a guest page directory or guest page tables. After this the component looks up from a nested page directory or nested page tables what host physical address corresponds to that guest physical address. [4] Since the hardware can handle both sets of address translations, the guest operating system kernels can change the mappings they see without the need for the VMM involvement [80]. The concept of hardware-assisted memory virtualization for paging can be seen in the Figure 20 on the next page.

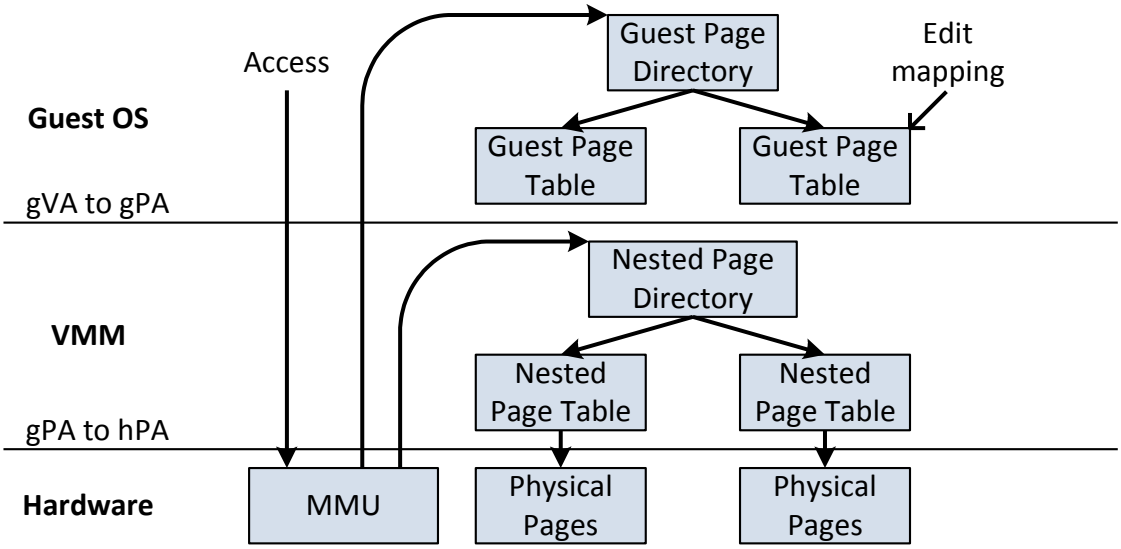

Figure 20. *The hardware-assisted virtualization extension for paging.*

The hardware-assisted CPU virtualization is required for the hardware-assisted memory virtualization. The segment register virtualization is done by the hardware-assisted CPU virtualization since it automatically saves the whole CPU state when the execution transfers from a VM to a VMM. Since the transferring the execution happens automatically, the VMMs and VMs can have their different address spaces and parts of the VMMs don't need to reside in a VMs' address spaces any more allowing the software in the VMs to use their whole address space. This would normally cause TLB flushes, but both NPT and RVI include a tagged TLB [80]. With the tagged TLB the TLB entries can be tagged to belong to a VMM or a VM and the TLB flushing is not needed when the execution is transferred between the VMs and the VMMs. In addition the NPT and the RVI can store data which indicates if the memory pages have been read or written to [52].

The ARM architecture has a similar memory virtualization extension. It also has a tagged TLB [118] and the memory management unit on it contains two sets of page directories and page tables similarly to the NPT and EPT [44][118].

The advantages of hardware-assisted virtualization are that the VMMs don't need to be involved when mappings are changed inside the VMs and that the VMMs can reside in different address spaces without the need for TLB flushes. Both of these improve the performance of the memory virtualization compared to the software-based memory virtualization methods [73]. However, there is one disadvantage with this method: TLB misses will be more costly and the memory access will take longer. This is because when a TLB entry cannot be found, the memory management unit needs to do the address translation which requires the memory management unit to walk through two sets of page directories and page tables [4]. The negative performance effects can be lessened by using larger page sizes allowing pages to contain more data and avoiding some of the page walks [73].

5.4.2 Management

The hypervisor can implement different kinds of memory management techniques to improve the memory usage on the host machine as memory is a limited resource. In addition the type 2 hypervisors have some special considerations as they have to share the memory with the operating system running alongside the hypervisor.

Depending how the type 2 hypervisor is implemented, a world switch might be required every time the execution switches between the hypervisor and the host operating system kernel. *A world switch* is a technique where the memory context is switched between the operating system kernel and the hypervisor. When the operating system kernel transfers the execution to the hypervisor, the memory address space is transferred from the operating system kernel's address space to the hypervisor's address space. At the same time the state of the CPU is stored so it can be restored later. [13] The goal is to provide protection for the operating system kernel and the hypervisor from each other. As mentioned previously, the swapping of a memory address space has a negative performance impact. The addition of swapping the CPU state makes the world switch even slower than traditional context switch [111]. This is why implementing operating system kernel and the hypervisor as separate pieces of software has worse performance than type 1 hypervisors. An alternative way of implementing the type 2 hypervisor is to implement it as part of the operating system kernel as a kernel module [13][60]. This requires some modifications or support from the operating system kernel but the world switch is not needed any more as the hypervisor and the operating system kernel work as a single piece of software [13][60]. In a way this turns the type 2 hypervisor into a type 1 hypervisor, but it still stays as the type 2 hypervisor as the operating system kernel can host normal applications and has functionality which is not for the virtualization purposes.

The memory architecture is a hierarchical structure and there are multiple levels of memory. There are the main memory components, which are separated from the CPUs, and then there are many smaller memory components, which are embedded into the CPUs. These smaller memory components are called as *caches* and the data stored into them is faster for the CPU to retrieve than the data stored into the main memory. The closer the cache is to the actual processing unit, the faster and smaller it is. When data is retrieved from the main memory, it is first transferred to the caches. Each cache retrieves a certain amount of data from the previous cache or from the main memory and the processing unit retrieves the data from the cache closest to the processing unit. [51] The Figure 21 on the next page shows an example of memory architecture.

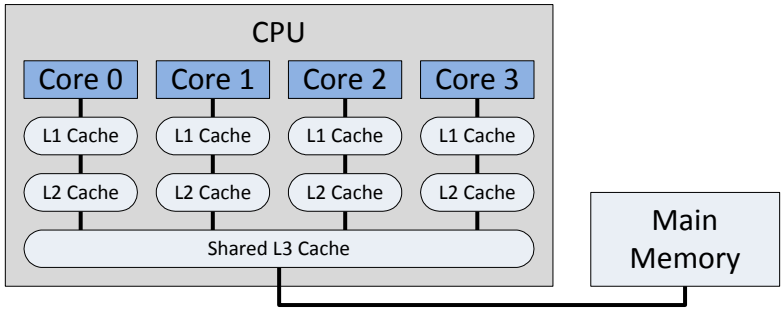

Figure 21. *An example of memory architecture. Modified from [51].*

The actual memory architecture varies between different implementations. However, they usually have some shared cache called as *a Last Level Cache* (LLC) which is the closest cache to the main memory. Typically the CPU manages these caches and there is no way for software to manage them. This can be problematic in virtualized environment where multiple VMs share the same hardware. As the LLC is shared, some VM processing memory intensive workload could cause data belonging to some other VM to be replaced in the LLC. The other VM has to transfer the data again from the main memory to the caches to access it. This can slow down the processing of the other VM. To solve this problem the Intel has developed a technology called as *Cache Allocation Technology* (CAT). With this technology it is possible to divide the LLC between VMs or applications in a way that they can't override the data from the parts of the LLC which aren't assigned to them. [51] The hypervisor can use this technology to provide separation for the memory the VMs use. Currently Intel is the only hardware provider providing this kind of technology.

When a VM is created, it is allocated by the hypervisor a minimum amount of memory it can use. In addition it can also be given a maximum amount of memory it can use. [9][117] This value can be anything from the minimum amount of memory to infinity. The minimum amount of memory is used to guarantee that the VM will always have at least some memory available to use. It is called *thin memory provisioning* when the VMs on the host machine are assigned more memory than is actually available on the physical hardware [73]. The assigned memory is not necessarily used. If the VMs use more memory than is available on the host machine, it is called as *memory overcommitting* [73]. Using more memory than is available on the host machine is not possible without using special techniques which allow the memory overcommitting. The hypervisor can implement any of the following techniques to allow memory overcommitting: *ballooning, page sharing, hypervisor swapping,* and *memory compression*. The detailed description of these techniques is presented in the Appendix B. The memory overcommitting allows the host memory to be used more efficiently as some of the VMs might be idle most of the time and have much memory which is not being used. With memory overcommitting techniques it is possible to get some of this memory into better use. [117] It is important to note that most of these techniques affect the performance of the

VMs running on the hypervisor as they require complex operations to allow the memory overcommitting.

5.5 Input / Output

Virtualizing the CPU and the memory is not enough. Computers receive and send data by using different kinds of input/output devices. Also these devices need to be virtualized for software to use. To understand the different I/O virtualization methods one needs to first understand how the I/O devices work in a computer. This involves knowing how the devices are controlled, how the devices inform the CPU of events, and how the data is transferred.

There are two ways how I/O devices can typically be controlled and used. These are called as *Memory-Mapped I/O* (MMIO) [1] and *Port-Mapped I/O* (PMIO). In the memory-mapped I/O the addresses to the registers that the device's CPU has are mapped to the memory. The device registers can be then read or be written to by reading or by writing to these memory addresses [1]. These addresses reside in the same address space as the software using the I/O devices. The port-mapped I/O is similar to the memory-mapped I/O. The difference is that the addresses reside in different address space and special CPU instructions must be used when reading from or writing to these addresses.

The I/O devices can signal the CPU of different kinds of events in two ways: They don't signal at all or they inform the CPU by using interrupts to interrupt the execution of the software currently running on the CPU. The first case where the I/O devices don't signal the CPU about events is called as *programmed I/O* and the second cased where the I/O devices use interrupts is called as *interrupt-driven I/O* [108]. In the programmed I/O the software running on the CPU has to keep reading the registers of the devices to notice the events. For example if some software sends data for a device to process, the software needs to wait and keep polling proper register to notice when the processing is done. When the processing is done, the software can continue its normal execution. This method is easy to implement but the down side is that the CPU is reserved the whole time until the device informs that it is done.

In the interrupt-driven I/O the software doesn't wait for the device to inform when it's done, but it continues the normal execution while the device is processing the data. Once the device is done with the processing, it sends an interrupt to the CPU which then transfers the execution to *an Interrupt Service Routine* (ISR). This interrupt handler does the required procedures what are needed after the data is processed. The interrupt delivery mechanism can be either in-band or out-of-band [126].

In *out-of-band* method the interrupt is delivered to the CPU by using separate lines that connect to pins on the CPU [1]. This method uses *Programmable Interrupt Controllers*

(PICs) to deliver the interrupts. The devices are connected with interrupt lines to the programmable interrupt controller and when they want to interrupt the CPU, they signal it by using one of such lines. Then the PIC signals the CPU by using its own interrupt line. The purpose of the PIC is to combine different sources of interrupts to one or more lines that are connected to the CPU and to deliver the interrupts to the CPU. The more advanced method uses *Advanced Programmable Interrupt Controllers* (APICs). There are two different kinds of APICs: local APICs and I/O APICs. Each CPU has one local APIC which services that CPU. The I/O devices are connected to the I/O APICs. Devices signal the interrupts to the I/O APIC they are connected to and the I/O APICs signal the interrupts to one of the local APICs. This local APIC signals the interrupt to the CPU it is servicing. [19] This is how interrupts are done in the x86 architecture, but the implementation of the interrupt delivery systems can be different in different CPU architectures.

In *in-band* method the interrupt is delivered by writing it to a certain reserved memory location. The delivered interrupts in the in-bound method are called as *Message Signalled Interrupts* (MSI) [1]. These memory locations point to the registers of the local APICs which allowing bypassing the I/O APICs [19].

The interrupts can be assigned priorities. When a CPU accepts an interrupt, it acknowledges it to the PIC or APIC which is connected to the CPU. After this the PIC or APIC won't try to interrupt the CPU with any same priority or lower priority interrupts. The ISR needs to send an *End Of Interrupt* (EOI) signal to the PIC or APIC to signal when it is done with handling the interrupt so that it can start receiving interrupts which are same or lower priority than the interrupt just handled. [52]

When the data is transferred to the devices, it's done by using the ISA instructions used to control the devices and by copying the data from the memory to the device registers. When the data is transferred from the devices to the memory, there are two ways this can be done: Either software uses the ISA instructions to copy move data from the device register to the memory or the device writes the data directly to the memory. The latter is called as *Direct Memory Access* (DMA) I/O. In DMA a device is given a predetermined memory region where it can write the data [108]. The device writes the data to that memory region and then signals the CPU when it's done. The MSI technology uses DMA to write the interrupts to memory locations. The I/O system of a computer can be roughly presented as in the Figure 22 on the next page when a simple PIC is used.

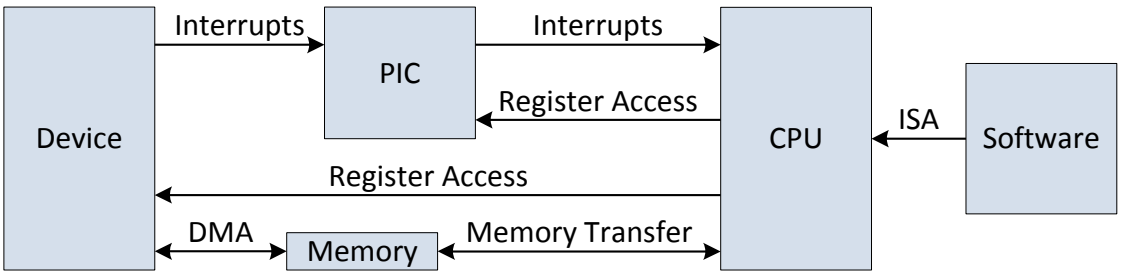

Figure 22. *I/O system of a computer. Modified from [29].*

5.5.1 Virtualization

To virtualize the I/O devices the device register access, the PIC/APIC register access, interrupts and the DMA need to be virtualized. The register access virtualization means that the MMIO and PMIO need to be virtualized. In full virtualization all of these aspects are emulated in software. When software in VM tries to use the MMIO or the PMIO, the CPU instructions are trapped and directed to the VMM so it can emulate the behaviour of them for the VM and to retain the control of the devices. The PMIO is easy to emulate since it uses special CPU instructions which can be trapped by using the CPU virtualization methods. The MMIO is not as easy as easy to emulate. The problem with MMIO is that it uses the same instructions as normal memory operations and the challenge is to differentiate between a normal memory access and a device access. One way of differentiating with the memory access and the device access is simply to not create gPA to hPA mappings for the MMIO addresses [61]. This will cause a page fault when software inside a VM tries to access the MMIO addresses allowing the VMM to notice the access to the MMIO addresses.

As different devices have different behaviour and registers, the VMM needs to emulate these for every device separately [124]. When the instructions to use MMIO or PMIO are trapped, they are directed to an I/O stack on the hypervisor [124][1]. In this I/O stack the I/O calls can be modified and managed by different pieces of software. One responsibility of the I/O stack is to virtualize the functionality of the device itself [1]. This involves multiplexing and scheduling the I/O calls for the physical device. Last part of the I/O stack is an actual device driver which interacts with the physical device. This architecture allows the emulated device to be different from the actual physical device [1]. The I/O calls are simply modified and directed to proper device driver in the I/O stack. The emulated device can also be called as *a device model* [1]. The I/O architecture is shown in the Figure 23 on the next page. The arrows represent the flows of information.

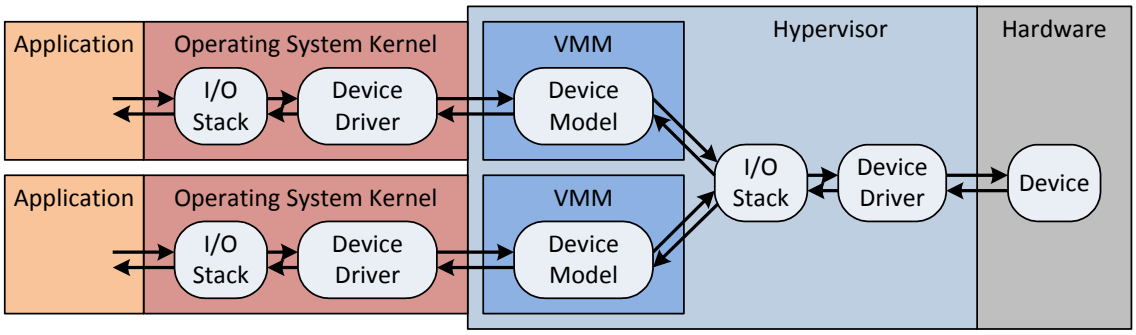

Figure 23. *The I/O architecture in full virtualization method.*

The interrupts need to be emulated as well. When the physical device signals an interrupt, it is delivered to the device driver inside the hypervisor. A notification of this interrupt is delivered to the correct VMM which emulates the device for the VM to which the interrupt was aimed for. The VMM then injects an emulated interrupt to the VM when the software in the VM is ready to receive interrupts. This is done by emulating the behaviour of different interrupt controllers. [1]

Address translation is also a major part of the I/O virtualization. When DMA-capable devices are used, they need to be told at what memory region they can write the data. Since the VMs use different address spaces than the host machine, the addresses used for the DMA transfers need to be translated to the host address space [124][1]. Otherwise the DMA-capable device could write to some memory locations that don't belong to the VM.

The advantage of the full virtualization is that the operating system kernels in the VMs can keep using the existing device drivers. These device drivers see an emulated version of the hardware so they can use it the way they would use the actual hardware. The disadvantages of the full virtualization are the complexity of the emulation and the negative performance effect it causes as the execution must be often transferred between the VM and the VMM in complex I/O operations. [1]

In paravirtualization the I/O device virtualization is done completely differently. In paravirtualization the existing hardware I/O technologies are not emulated, but instead a new method to access the existing I/O devices is created. The old device drivers are replaced with new *split device drivers.* As the name tells, these new device drivers are split into two different parts: *front-end drivers* and *back-end drivers* [67][86]. The front-end driver is the part of the device driver which is installed into the operating system kernels running in the VMs. The back-end driver is the part of the device driver which is installed into the VMMs.

The VMMs provide hypercalls in their hypercall interfaces which can be used to connect the back-end and the front-end drivers and to communicate between them. Typical way of implementing the communication is by shared memory and buffer rings [9][95].

The buffer rings contain descriptors pointing to shared memory areas which contain the data for the I/O requests and responses. The VMM validates all the I/O requests. [9] The memory is shared simply for performance reasons [42] as then the data doesn't need to be copied from one memory location to another. The I/O calls can be directed from the back-end driver to the I/O stack in the hypervisor. The native device driver is part of this I/O stack the same way it is in the full virtualization [92][86]. It is important to note the since there is no need to emulate the hardware, the whole I/O stack along with the native device drivers can be part of the back-end driver. This way there will be only one back-end driver which handles the I/O calls from all the VMs [56].

Interrupts in paravirtualization are typically done with events or callbacks [9][1]. When the physical device sends an interrupt, the back-end driver reacts to it and then sends an event to the front-end driver in a VM or uses the front-end driver's callback function. One of the I/O architectures for paravirtualization method is shown in the Figure 24.

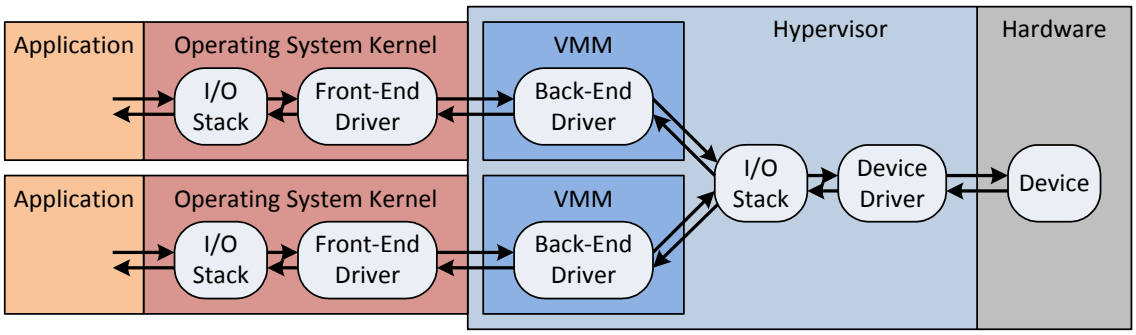

Figure 24. *One of the I/O architectures in paravirtualization method.*

The advantage of the paravirtualization method is better I/O performance compared to the full virtualization method as the hardware is not needed to be emulated and the I/O calls can be done with API calls which are optimized for a virtualized environment [73][1]. The disadvantage of this method is that it requires completely new device drivers to be developed [1]. This can be more acceptable than CPU or memory paravirtualization since the operating system kernels can stay unmodified [73].

Both of these software-based virtualization methods have a one common advantage: they both can encapsulate the state of the virtual devices since the virtualization is done in the software [1]. The VMMs know what I/O calls have been made and at what stage they are in the execution. This provides some benefits which are discussed more in the Chapter 5.6 which talks about how the hypervisor can use encapsulation to provide some new functionality.

Since the I/O operations involve many different hardware components, there are many modifications required to the hardware to support hardware-assisted virtualization. Both Intel and AMD have developed a new hardware component for their x86 architecture which can handle the address translations when using DMA-capable devices. This new

component is called as *Input/Output Memory Management Unit* (IOMMU). Hardware containing the IOMMU is called as VT-d enabled by Intel [1]. The behaviour of the IOMMU is similar to the traditional MMU. It contains page directories and tables which are used to translate the guest physical memory addresses given by the software in VMs to the host physical addresses [53][5]. The IOMMU contains a cache to look up already translated addresses as well. This cache is called as *Input/Output Translation Lookaside Buffer* (IOTLB) [53]. This way the VMMs don't need to do the translation when VMs use DMA-capable device. The translation is done by the hardware when a device tries to write data into the memory. The address mapping is done by the VMMs.

Additionally IOMMU provides new protection domains. *A protection domain* defines an isolated environment which has part of the host physical memory. Each domain can be assigned one or more I/O devices. The I/O devices can write to the memory regions of the domains which they belong to, but they can't write to memory regions which belong to other domains. The protection domains simply work by preventing the memory writes to addresses which are not assigned to the devices in the page tables. [53][5]

The IOMMU also changes the message format of the MSIs. Typically the address where the MSI is written contains interrupt attributes such as the destination processor, delivery mode, etc. Because the MSIs are issued by the devices and not by the VMs, the protection domains won't work [1]. This is why the MSI messages are modified to contain message identifiers instead of the interrupt attributes [53][5]. These message identifiers are then used to map the MSIs to correct physical addresses by the IOMMU. When the message identifier is translated to the host physical address, the tables used to translation contain the interrupt attributes. [1]

If guest operating systems use the IOMMU themselves for memory protection, the VMM needs to emulate the IOMMU as well. The IOMMU emulation can be done in the software using similar methods which are used to emulate traditional MMU in software [1]. Additionally the IOMMU provides two stage address translation which can be used as a hardware-based virtualization method similar to the hardware-based memory virtualization [53].

IOMMU solves the address translation in hardware, but there are still the interrupts and the register access that needs to be virtualized. The hardware-assisted CPU virtualization technologies can be used to virtualize the interrupt delivery. With the hardware-assisted CPU virtualization the VMMs can inject interrupts to the control structures which are loaded when the execution is transferred to VMs [80]. This way the VMMs don't need to emulate the delivery of the interrupts and the interrupts are handled in the VMs when they are executing. One problem still exists: the VMMs are needed to inject the interrupts to the VMs. To solve this problem Intel has developed a technology called as *APIC virtualization* (APICv). AMD has announced similar technology called as *Advanced Virtual Interrupt Controller* (AVIC) [46] and there are some references to it in

AMD's IOMMU specification [5], but as of yet it has not been implemented in any CPU architecture. Because of this, only the APICv is discussed here.

The APICv requires hardware-assisted CPU virtualization to work. This is why it is part of the newer CPUs which have the VT-x virtualization technology. In APICv there are virtual APIC-tables [52]. These tables are used to store the state of local APICs for each VM. When a software in a VM tries to read or write the APIC's registers, the reads or writes are directed to the virtual APIC-table, and when an interrupt for a VM arrives to a local APIC, it is directed to the virtual APIC-table as well [79]. If the VM is currently running, it can process the interrupts immediately [52], but if the VM is not running, the interrupts can be stored in the APIC-table as pending. This removes the need to transfer the execution to a hypervisor so it can direct the interrupt to a correct VMM and VM [79]. When the CPU, to which the APIC belongs to, receives a notification vector, the pending interrupts, which are marked in the notification vector, are processed. This is called as *posted-interrupt processing* [52]. This allows a VMM to set the interrupts belonging to a VM to be processed when it sets it to execute on the CPU. APICv reduces the need to transfer the execution to the VMM allowing better performance [59].

Lastly the MMIO and the PMIO need to be virtualized as they are what allow the access to the device registers. The access to these can be virtualized with the hardware-assisted CPU and memory virtualization methods. This can also be called as *device passthrough* [73]. However, since many I/O devices are designed to be used only by one device driver, the functionality of the I/O devices needs to be virtualized as well. [86] If this is not done, the access to the I/O device can be allowed only for a one VM. One technology developed to solve the problem of sharing I/O devices with multiple VMs is called as *Single Root Input/Output Virtualization* (SR-IOV). SR-IOV is a specification made by Peripheral Component Interconnect Special Interest Group (PCI-SIG) and it describes how I/O devices, which use *Peripheral Component Interconnect* (PCI) technology to communicate with the rest of the machine, can be shared with multiple VMs [29]. It basically provides an interface for multiple VMs to use the device at the same time so the functionality of the I/O stack can be transferred to the device itself.

A typical PCI device has a configuration space and multiple functions which can be read or be written with the MMIO or PMIO. The configuration space contains the configuration of the device and the functions are used to use the device [29]. The messages used in PCI communication have *a Requester IDentification* (RID) which can be used to determine who sent the message. Each function has a unique RID which can be used to identify it [29]. In SR-IOV the functions are divided into two categories: *physical functions* (PFs) and *virtual functions* (VFs). Physical functions are able to configure and manage the configuration space and change the device's behaviour. They can also allow using all of the functions which the device has available. The virtual functions are simpler and they only contain resources required for the data transfers and they have limited configuration abilities. [86]

Using the physical functions and the virtual functions require separate drivers: one for the physical functions and one for the virtual functions. The hypervisor uses the physical functions to configure and manage the configuration space of the device. It configures separate virtual functions for VMs to use. It can also be used to use the device when a VM is wanted to use software-based virtualization methods for the I/O virtualization. [86] SR-IOV requires hardware-assisted CPU, memory and DMA virtualization to work. The hardware-assisted CPU and memory virtualization methods are used to allow direct access for the VMs to these virtual functions and an IOMMU is used to translate the addresses in I/O calls and data transfers. The RIDs in the PCI messages are used for the address translations [29].

The advantages of SR-IOV are that it allows software running in VMs to use the I/O devices directly without the need for the VMMs to intervene allowing native speed for the I/O calls and at the same time there is less code required to implement in the VMMs. The disadvantage is that the device state isn't captured inside the VMM since the VMM can't see at what I/O calls are made. This is the problem of device passthrough and it has some negative effects which are explained in the Chapter 5.6.

To demonstrate the effectiveness of the hardware-assisted virtualization methods, the Figure 25 shows two different virtualization cases.

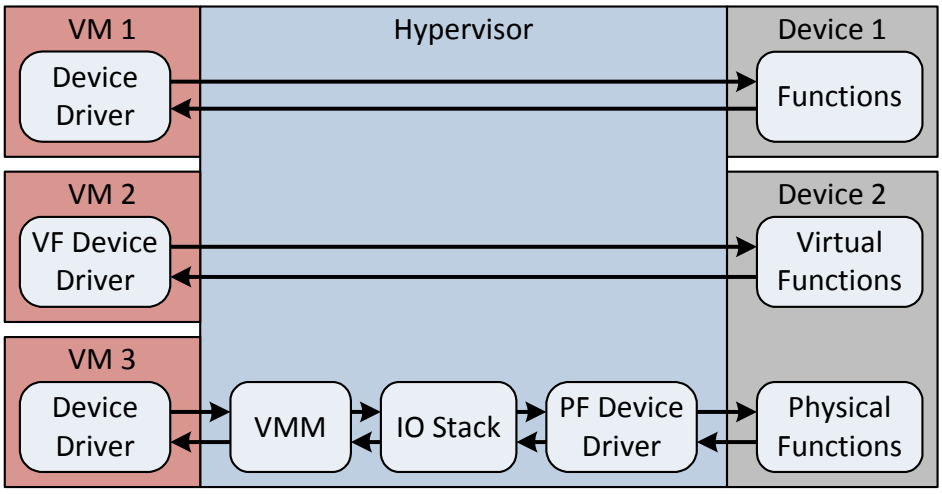

Figure 25. *Different I/O virtualization cases.*

In the first case the VM 1 uses an IOMMU, APICv, and hardware-assisted CPU and memory virtualization methods. This allows the VM to use the device directly, but since the device 1 is not SR-IOV enabled, no other VM can use the device. In the second case the VM 2 also uses an IOMMU, APICv, and hardware-assisted CPU and memory virtualization methods. The difference to the first case is that the VM 2 uses driver designed to use virtual functions so it can use the SR-IOV enabled device 2. This allows it to use the device directly and still share it with the VM 3 which uses the device with full virtualization methods.

The ARM CPU architecture implements many similar hardware-assisted virtualization methods for the I/O. It implements a similar hardware-component to the IOMMU, but instead of calling it as an IOMMU, it is called as *System Memory Management Unit* (SMMU). It implements two staged address translation for the DMA capable devices in a similar manner to the IOMMU. It can also be used to protect software running in some VM from corrupting the memory of software running in another VM. [75]

The interrupt virtualization in the ARM CPU architecture is quite different from the interrupt virtualization in the x86 architecture. In the ARM architecture there is a one interrupt controller called as *Generic Interrupt Controller* (GIP). This interrupt controller collects all the interrupts and then forwards them to correct CPUs by using CPU interfaces. The ARM's virtualization extension adds *virtual CPU interfaces* to this controller. These virtual CPU interfaces can be assigned to different VMs. When an interrupt arrives, it is forwarded to a VMM which can do the correct translations and generate a virtual interrupt which is sent back to the GIP. The GIP then sends the virtual interrupt to a correct virtual CPU interface which interrupts the correct VM. [8] How the interrupt virtualization works in the ARM architecture can be seen from the Figure 26. The parts belonging to the I/O virtualization are marked with black.

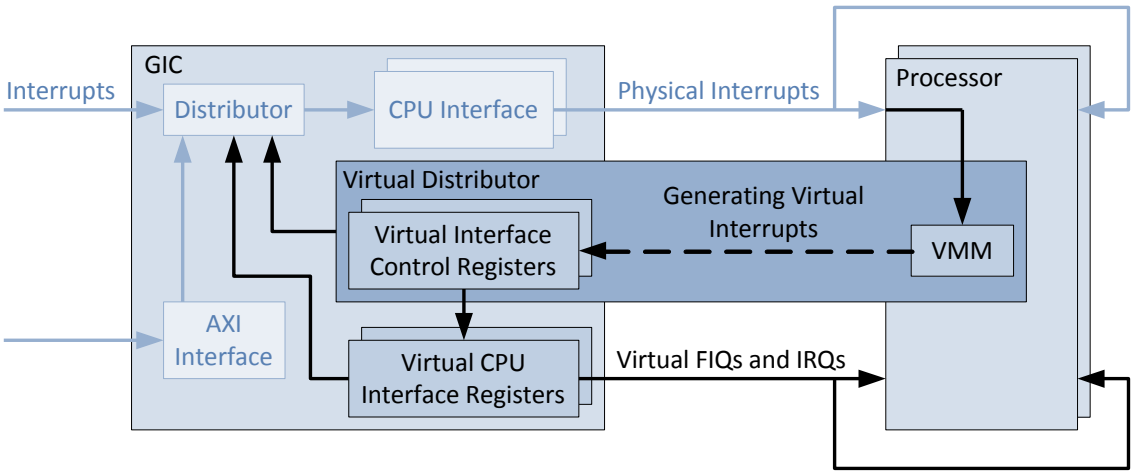

Figure 26. *The interrupt virtualization in the ARM CPU architecture [8].*

This method of interrupt virtualization requires the VMM to generate the virtual interrupts from the physical interrupts requiring some VMM intervention. However, the GIC doesn't need to be emulated and the VMs can use it without any VMM intervention allowing better performance.

5.5.2 Management

There are few different ways the hypervisors can use the physical I/O devices. The type 1 hypervisors can either implement their own device drivers or they can assign privileged VMs which can use the I/O devices directly [1]. In the second case the native device drivers are installed in privileged VMs along with the device emulators or para-

virtual back-end device drivers. The privileged VMs can also be called as *driver domains*. The privileged VMs run some existing operating systems. The hypervisor provides means to communicate between the privileged VMs and the non-privileged VMs. Using privileged VMs allows the hypervisor to use the pre-existing native device drivers which have been developed for the operating system kernel running in the privileged VM [1]. This also provides stronger fault resistance for malfunctioning device drivers. If a device driver malfunctions and crashes the kernel it's running on, only the operating system kernel running in the privileged VM will crash and the hypervisor itself stays unaffected. In addition the privileged VMs can be used to provide protection for the device drivers from each other. A crash in one of the privileged VMs won't crash the rest of the privileged VMs. The disadvantage is that the performance is slightly worse than if the hypervisor implemented its own device drivers as now the execution must be transferred between the hypervisor, the non-privileged VMs and the privileged VMs [1]. If the hypervisor implements its own device drivers, it can cause portability issues if the hypervisor doesn't have the device drivers for the devices the host machine has. At the same time the code base that the hypervisor must implement becomes larger. [1] The type 2 hypervisors use the native drivers provided by the host guest operating system kernel they share the hardware with. This resembles the method where the type 1 hypervisor uses existing operating system kernel for the native drives. However, on type 2 hypervisor this doesn't provide protection from malfunctioning device drivers as the operating system kernel and the hypervisor both run in the supervisor mode. The type 2 hypervisors can implement functionality which allows communication between the host operating system and the guest operating systems [97]. One example of this kind of functionality is the clipboard functionality which allows copying and pasting data. User could copy something in the host operating system and the paste it inside the guest operating system.

The hypervisor manages which devices the VMs can see and use. It also manages the passthrough of devices. If the device doesn't have SR-IOV, it can only be passed through to a one VM. Additionally if the device can use DMA, IOMMU is required to protect the memory of the hypervisor and other VMs [99]. With the type 2 hypervisors it might be impossible to use device passthrough. The reason for this is that the host operating system generally owns the devices and the hypervisor can't manage them [1]. However, if the type 2 hypervisor is implemented as a kernel module, it is possible to allow the device passthrough as the hypervisor and the host operating system kernel can work more closely together.

The I/O management consists of things like partitioning and sharing the I/O resources between multiple VMs. This involves scheduling the I/O operations [9][10] and limiting the usage of the I/O device [109]. The limiting can be done for example with bandwidth limiting, with I/O operations limiting, or by giving shares to the VMs to use. These functions are often called as QoS functions. The QoS functions can be provided by the

hypervisor, the host operating system kernel, or the device itself and the implementation of the QoS functions can vary a lot and be dependant of the device for which they are implemented for. This is why these QoS functions are not explained here. The guest operating systems schedule the I/O as well [10]. Some schedulers schedule based on the state of the I/O device for which they are scheduling for. These schedulers might not work properly in virtualized environment as the state of the virtual I/O device doesn't necessarily mirror the actual state of the I/O device. For example some I/O device might be congested yet the virtual I/O device is not. The combination of the guest I/O schedulers and the hypervisors I/O schedulers can affect the performance and proper schedulers should be selected based on the workloads [10].

The I/O management in the hypervisor can also provide other functionality as well [9]. For the network management the hypervisors can for example implement virtual switches to provide communication between the VMs it hosts. The hypervisor connects virtual network interface cards (NICs) inside the VMMs to the virtual switches. The virtual switches can also be connected to the physical NICs on the host machine. [99] In addition to these features the hypervisor can implement any other network features it wants. It can for example implement a software firewall to control the traffic that goes through the virtual switches [9][99].

The VMMs provide virtual block devices for the VMs and the guest operating systems to use. The blocks written to these virtual devices can be stored into a file or they can be written directly to a physical storage device [97]. For this reason the hypervisor should implement a file system and be able to use some block virtualization technologies. If the blocks are written to a file, the file acts like it is a storage device. The size of the emulated storage device can be controlled by changing the size of the file. If the blocks are written directly to a physical storage device, the storage device should be divided into logical volumes. Each VM will get its own logical volume where it can write data. For example a LVM can be used for this purpose. If networked storage devices are used, the hypervisor should implement a file system which can use these devices. Writing data blocks to a file typically provides worse performance than writing the data blocks directly to a storage device [130]. The reason for this is the extra layer added by the file system to the storage access stack.

Timekeeping in hardware virtualized environment requires special attention. In virtualized environment there are three different concepts of time: real time, virtual time and wall-clock time. The real time is counted from the starting of the host machine. The virtual time is counted from the start of a VM and it is counted only when the VM is executed. The wall-clock time is the absolute time. As the hardware clocks are emulated for the VMs, they run in virtual time and the guest operating systems use the virtual time to make their decisions. [9] However, sometime software requires real time for its functionality to make proper calculations and users of the VMs often require the VMs to have proper wall-clock time. In paravirtualization the hypervisor can offer the guest

operating systems the real time to use and it can even be used to make more intelligent scheduling decisions in the guest operating systems [22].

Operating systems typically keep real time in one of two ways: either by tick counting or by tickles timekeeping. In *tick counting* the hardware is set to interrupt the operating system periodically. The hardware has clocks which generate interrupts called as ticks. The operating system calculates the time by counting the ticks and by knowing the frequency of the ticks. In virtualization the VM might not be executing while the ticks happen. The guest operating system will lack behind as it can't process the ticks. The hypervisor and the VMM will collect a backlog of clock ticks which the VMM should inject to the VM once it is its turn to execute. The tick counting relies on the ticks arriving periodically so they cannot be processed immediately in one chuck. The hypervisor can try to solve this problem by increasing the speed it injects the interrupts to the VM or simply just ignoring the backlog if it grows too big. Either way the timekeeping in the VM will not be perfectly accurate. Even if the VM was executing and receiving ticks, it might still lack behind in the timekeeping. The reason for this is that the hypervisor and the VMM might have to intervene when injecting the ticks causing the interrupt frequency to be smaller than what it really is. Tick counting can also cause scalability issues. Even if the software in VMs doesn't have anything to do, the VMs must be assigned some CPU time so they can process the ticks. The assignment of CPU time could be avoided if the timekeeping was done by using some other method. Some of the problems of tick counting can be tried to be avoided by changing the frequency the guest operating system uses for ticks and by avoiding overcommitting the CPU of the host machine so there is enough CPU time to allow processing the ticks for each VM. Any memory management methods which affect performance should also be avoided. [77][115]

In *tickles timekeeping* a hardware device keeps a count of time units since the system has been booted. This is done by having a hardware counter and a clock which runs at constant known frequency. In tickles timekeeping the operating system simply asks for the value of the counter and deduces the time from that value and from the known frequency. This method of timekeeping is much better for virtualized environment as a guest operating system can ask for the time when it needs it and no constant ticks are needed to be passed to the operating system. The hypervisors needs to know if the guest operating system uses tickles timekeeping. Otherwise it has to assume it uses tick counting as a timekeeping method. Paravirtualized device driver can be used to inform the hypervisor that the guest operating system uses tickles timekeeping. [115]

Machines initialize the wall-clock time either from a battery-backed real-time clock or by querying it from a network during the booting phase. If the timekeeping is not accurate, the kept wall-clock time can drift apart from the real wall-clock time. The hypervisor should keep an accurate wall-clock time either by using the hardware clock or by querying the time from a network. The guest operating systems can synchronize to the

wall-clock time by either using paravirtualized device drives, which asks the wall-clock time from the hypervisor, or by querying the time themselves directly from a network. [115]

5.6 Encapsulation

The hypervisor is in control of the hardware and any attempts to manage and configure the hardware should be possible only through the hypervisor. This allows the hypervisor to know the exact state of the VMs and their virtual hardware. The state of the VMs is encapsulated and the software running in the VMs is decoupled from the actual physical hardware. This property allows the hypervisor to provide some functionality which wouldn't be possible in environments without virtualization.

Since the hypervisor is aware of the whole state of a VM, it can save it to a storage device. The state of a VM includes things like the contents of the memory, the contents of the VM's storage devices and the status of the hardware devices the VM uses [63]. The state is typically saved on a storage device as a single file or a folder [97]. It is also possible that the data in the VM's storage devices is directly stored on its own volume in a block format and only the metadata linked to this volume is stored in a file [123]. There are two different approaches for saving the state: one can save the state without the contents of the memory and the status of the hardware devices, or the contents of the memory and the status of the hardware devices can be included.

In the first approach the saved state is called as *a virtual machine image* [97]. The VM image can be used to restore back the VM to run on the hypervisor. The restored VM acts like it is booted up after a shutdown [63]. This feature can be used to create images which have all the necessary tools and software already installed in the VM. Whenever a new VM is required, the image can be used to launch a new VM which is already properly configured. To create the VM image safely, software that could alter the data stored in the VM's storage device should be set to a state where the VM image can be created safely [63]. One way of doing this is by shutting down the VM for the duration when the VM's image is being created. Another way is to have special software installed on the guest operating system which sets the file system and any other software in the operating system into safe state before making the VM image [63].

In the second approach the contents of the memory and the status of the hardware devices are saved as well. The saved state in this case is called as *a virtual machine snapshot* [97]. With the VM snapshot the contents of the memory can be restored to the VM as well. This allows restoring the VM to the exact same state when the snapshot was taken and the VM doesn't require booting up. When a snapshot is being made, the whole VM must be halted so that the memory can be written to a file [63]. If the VM was running, it could alter the memory in middle of the process and the saved state wouldn't be consistent. The saved state should be stored in specific point in time. Some-

times snapshots can create a hierarchical structure. When a snapshot is created, only the data which has been modified from the last snapshot on the VM's storage device are stored [63]. This saves storage space and speeds up the snapshotting process as the data from old snapshot is still valid and only the modified data needs to be stored.

The VM images and snapshots can be used to implement fault recovery features in the hypervisor. The hypervisor could restore a VM back to run if there is a failure with it. Similarly multiple hypervisors could communicate with each other and in case one of the physical hosts becomes faulty, some other hypervisor could restore the VMs from that host assuming that the VM images and snapshots were stored in network storage. [76] The hypervisor could be used to provide fault tolerance against faulty devices as well. If a physical device becomes faulty on the host machine, the hypervisor could map the virtual devices which use the faulty physical devices to another physical device which is able to provide the same functionality. [124]

Another new feature allowed by the state encapsulation is called as *live migration*. The live migration is a technology which allows transferring a working VM from one physical host to another with a minimal downtime for the VM. This requires transferring all the necessary parts of the state. The whole state is not necessarily required to be transferred if the storage, which the VM uses, is in networked storage. In this case the data in storage doesn't need to be transferred. The live migration can consist of three different phases: a push phase, a stop-and-copy phase, and a pull phase. In *the push phase* the data is copied from a source VM's state to the destination VM while the VM is running. Data which is changed during the push phase must be resent to make the changes effective on the transferred copy. In *the stop-and-copy phase* the VM is stopped and the data is copied from the source to the destination. Changed data is not a problem as the VM is not running. In *the pull phase* the new VM executes at the destination and every time it tries to access data that has not yet been copied, the data is copied from the source. Any combination of these phases can be part of the implementation of the live migration functionality. For example the live migration can consists only of the stop-and-copy phase in which case the VM has long downtime while the data is being copied to the destination. [18]

A one way of doing the live migration is shown in the Figure 27 on the next page. The live migration consists of five different stages: pre-migration, reservation, iterative pre-copy, stop-and-copy, commitment, and activation. In the pre-migration stage the VM, which will be migrated, is simply running and operating normally. In this stage it is also possible to select the target hypervisor where the VM will be migrated when needed. The target hypervisor should maintain enough free resources for the VM in case it is migrated. In the reservation stage the available resources are confirmed from the target hypervisor and an empty VM reserved in the target hypervisor. If there are not enough resources available, the live migration will not be done. [18]

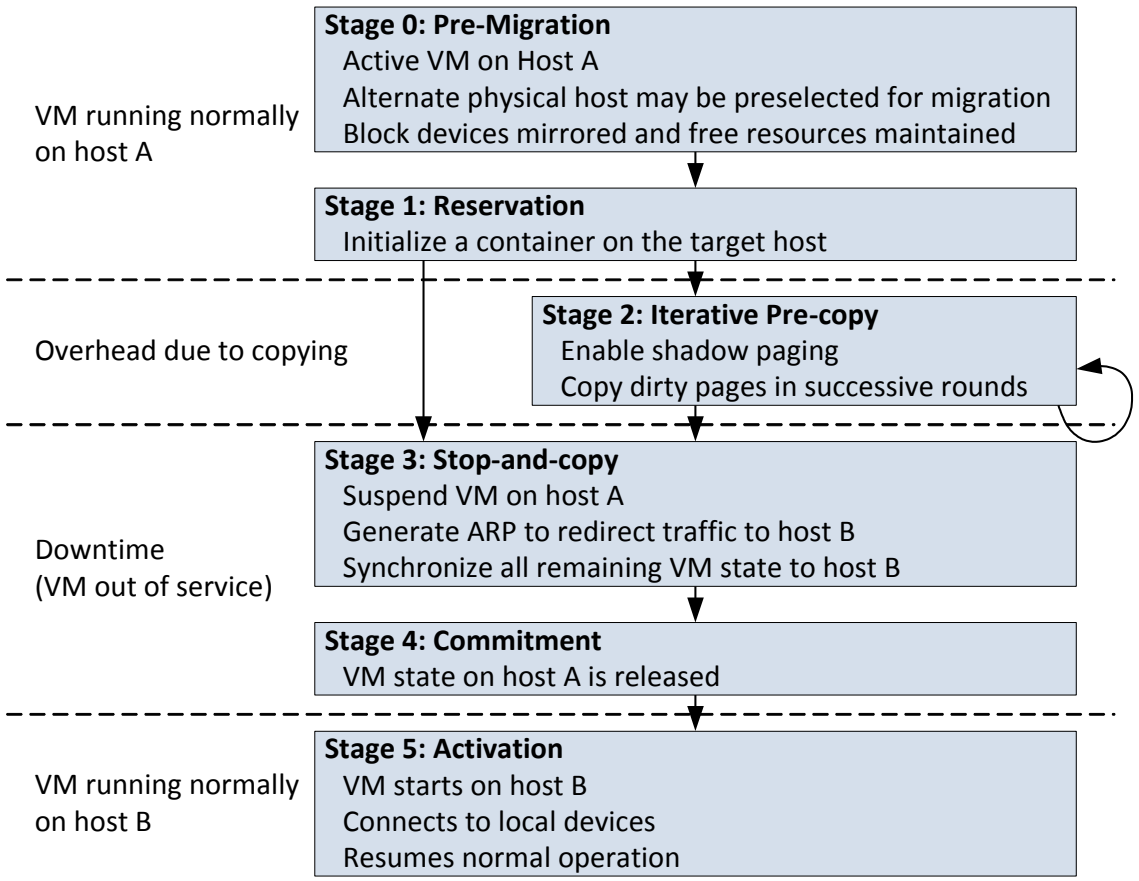

Figure 27. *Stages of live migration [18].*

In the iterative pre-copy stage the data is copied to the target hypervisor. In this stage the VM is kept running and the data is copied to the target hypervisor in iterations. If some data is changed during this face, it is marked somehow and then copied again on next iteration. For memory copying this requires that all the edits to the memory need to be trapped to the hypervisor. This can be done by setting the memory pages as read only. When software in VM tries to edit a memory page, the edit is trapped and the edited page is marked as a dirty page in the hypervisor and then the memory page is edited. This will cause negative performance impact as all the memory edits must be trapped by the VMMs on which the VMs run. As the hardware-assisted memory virtualization technologies support marking edited memory pages, this can be used to minimize the performance loss as the VMMs don't need to intervene every time when a memory page is edited. The iterative pre-copy stage ends after some conditions are fulfilled. In the stop-and-copy stage any network communications are redirected to the new VM on the target hypervisor and the source VM is suspended. In this stage the last pieces of data, which weren't transferred in the iterative pre-copy stage, are transferred to the target hypervisor. The downtime for the VM can be as low as 60ms, but the downtime depends on the amount of data required to be transferred, the bandwidth of the network connections, and the behaviour of the applications the VM hosts. In the commitment stage the new VM is made to be the primary VM and the source VM can be discarded.

Lastly in the activation stage the devices the VM uses are mapped to the devices on the new hypervisor and the VM is set to run. [18][81]

The encapsulation allows also record and replay capability for the VMs. *The record capability* is used to save the state of the VM as it executes so that the state can be repeated by using *the replay capability*. To save the state, the hypervisor needs to store events that happen on the hardware and the time when they happen. However, it is not necessary to save everything that happens on the hardware. Much of the functionality is deterministic and the next state of the VM can be determined from the previous state of the VM. The events caused by deterministic functionality don't need to be stored. Still some of the functionality is not deterministic. For example any inputs from I/O devices cannot be determined by examining the functionality of the hardware and the previous states of the VM. In these cases the non-deterministic events are required to be stored in a log. The entry in the log contains the time when the non-deterministic event happened and any data that is related to that event. The time can be indicated for example by counting instructions executed by the VM and noting after which instruction the non-deterministic event happened. [66][133] The starting state of the VM should be stored as well so that any deterministic functions can be replayed correctly.

To replay the states of the VM, the execution should be started from the stored starting state. The VM is executed normally until the first logged non-deterministic event happens. The hypervisor executes this non-deterministic event at the correct time during the VMs execution. As the devices work differently, the record and replay functionality should be aware how the devices work in order to correctly record and replay the required data associated with the non-deterministic events. [66][133]

The hypervisor can also monitor the resource usage of the VMs and provide information of the resource usage. To get the proper measurement of the host machine's resource usage, the monitoring must be done in the hypervisor. If the monitoring is done inside a VM, the monitoring can give approximately correct results for the resource usage of the virtualized devices and not for the actual physical devices [115].

All these features require that the whole state of the VM is encapsulated and the software is fully decoupled from the hardware. The hypervisor needs to know the state of the VMs and it should be able to control the operation of the VMs if necessary. The device passthrough feature breaks the encapsulation property by allowing the software inside the VMs to directly use the I/O devices. The VMM can't monitor the state of the devices and the hypervisor isn't able to formulate the device state and control the devices. This prevents the hypervisor from implementing the features presented in this chapter [124]. The devices which are passed through to the VMs should be first disconnected to be able to use these features.

6. OPERATING SYSTEM –LEVEL VIRTUALIZATION

Operating system (OS) –level virtualization is another technology comparable to the hardware virtualization. While the hardware virtualization aims to virtualize the ISA and the hardware, the operating system –level virtualization aims to virtualize *the Application Binary Interface* (ABI) and the operating system kernel [105][128]. The ABI is an interface provided by on operating system kernel. It provides system calls which can be used to manage the hardware through the operating system kernel. User ISA is also part of the ABI. [103] The OS-level virtualization provides emulated versions of the operating system kernel. An operating system contains operating system kernel which is bundled up with application and libraries. Since the OS-level virtualization virtualizes only the operating system kernel, multiple operating systems can be run on a same host machine assuming that they all use the same operating system kernel [105]. This allows running different distributions of Linux to be run on same virtualized operating system kernel as the distributions share the same kernel. However, running Windows and Linux on same host machine is not possible.

In OS-level virtualization the emulated operating system kernels can be called as virtual machines [105], *containers*, or *zones* [91]. This thesis uses the term container. The purpose of the containers is to create isolated environments for applications to run in. Any application running in a one container should not be able to see or manipulate the resources belonging to some other application running in a different container. Each of the containers can contain multiple applications or just a single application. This is similar to what hardware virtualization does. However, in hardware virtualization each isolated environment has its own operating system kernel. One of the containers is designated as *a host container* which contains all the management tools and which has privileged access to the operating system kernel [105]. In OS-level virtualization a single operating system kernel is shared with all the isolated environments [105]. The operating system kernel must be modified or it should already support creating isolated environments. The OS-level virtualization offers better performance than hardware virtualization as it removes the need for the guest operating system kernels. Difference between hardware virtualization and OS-level virtualization is shown in the Figure 28 on the next page.

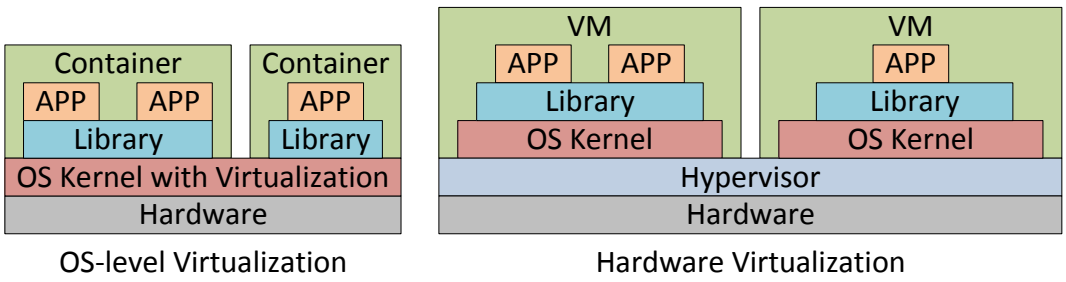

Figure 28. *The difference between OS-level and hardware virtualization.*

In OS-level virtualization the resource isolation can be done in one of two different ways: by using namespaces or by using context IDs [91]. The term namespace was already mentioned in the storage virtualization chapter when discussing about file systems. The file systems contain tables which are connected to each other by addresses. *A namespace* can be considered as a table which contains data and which is accessed from some other table. This same principle can be extended to other resources the operating system kernel has. Tables are created which contain addresses to other tables. Each table is a container specific table and contains only the data belonging to that container. The isolation is achieved by having the data in separate tables and restricting the containers' access to only to the tables that belong to them. [105] The other method is using *the context IDs*. Instead of creating a new namespace for each container, the existing data structures are modified to contain a context ID [91]. Each container is assigned a unique context ID which is checked when a container tries to access data. The operating system kernel isolates the containers from each other by managing the access to the data structures by matching the context IDs. Using the namespaces or context IDs can be categorized as *contextualization*. Controlling the container access based on these methods can be called as *filtering* [105].

The shared resources which require isolation between containers are: file systems, process IDs, Inter-Process Communication (IPC), I/O devices, CPU, and memory [127]. As the file systems already use namespaces by design, it is natural to use the namespaces to isolate the containers from each other. Each container is given its own directory which works as its own file system namespace. When software inside a container executes, the root directory of the file system is changed to the directory which belongs to the container. [127] Many operating system kernels provide functionality which can be used for this operation. Examples of this kind of functionality are the Linux kernel's chroot() and pivot_root() system calls. The chroot() system call has the problem that any software that has proper privileges inside the container can use the chroot() to change the root directory again and break the isolation. Pivot_root() system call doesn't have this problem and it is used in many of the OS-level virtualization solutions. [91]

Process IDs (PIDs) are stored in tables inside the kernel. Each process has its own unique ID which other processes can use to identify the process [127]. Both namespaces and context IDs can be used to virtualize the process table. With context IDs processes

can’t have same ID as they reside in the same PID table. With namespaces each container can have its own PID table and the IDs can be reused. [105]

Processes communicate with each other by using concepts like shared-memory segments, semaphores, and messages [127]. These reside as objects inside the kernel. Processes inside a one container should not be allowed to communicate with the processes inside other containers. For this purpose the namespaces or context IDs can be used to separate the IPC objects. [91]

Similarly to the hypervisors, the operating system kernel in OS-level virtualization has to implement methods to limit the resources the containers can use. This involves things like CPU scheduling, limiting the amount of memory containers can use, scheduling for I/O devices, and QoS for the I/O devices. The scheduling algorithm should be selected based on the use case. [105][91] In some cases the schedulers can take into account the containers and their priorities [127] and sometimes the scheduler can just schedule based on all the processes. Storage and network virtualization technologies can also be used in the operating system kernel. For the memory isolation the operating system kernel can use the hardware memory virtualization technologies though the nested page tables are not needed as there are only two levels of protection needed: the operating system kernel and the applications in the containers. The access to the I/O devices must also be restricted and multiplexed similarly to the hardware virtualization. Containers should not be able to create device nodes in the kernel unless the devices are purely virtual, stateless, or user namespace-aware [91]. Purely virtual devices are designed for virtualization and each container can have its own version of the virtual device. Stateless devices do not contain state so using these devices is safe. User namespace-aware devices can identify the container and act accordingly so they are safe to use as well. For example in Linux kernel *control groups* (cgroups) can be used to control and limit resources for containers. [91]

Sharing common data with multiple containers is one of the key components of the OS-level virtualization as the whole kernel is shared between the containers. If some container tries to modify data that is shared with multiple containers, copy-on-write feature can be used and new copy of the data can be made for the container. After this the container can modify and use the copied data [105].

While in the OS-level virtualization the kernel is required to implement similar management methods to that the hypervisors implement, the kernel can also implement the extra functionality allowed by the encapsulation. The kernel can implement snapshotting and live migration for easier container management. [127] Using namespaces for isolation allows easier implementation of these as same PIDs can be used for different containers. With context IDs the PIDs would need to be properly changed after a snapshot restore or a live migration if some PID is already in use. [105]

The benefit of OS-level virtualization compared to the hardware virtualization is the better performance. However, it provides less isolation than hardware virtualization [128]. Other disadvantage is that the kernel has to be shared between the containers. Operating system kernels are complicated which makes it harder to implement isolation between the containers. Any modifications to the kernel or security issues within the kernel can cause security issues with all the containers. In addition if some software would require special kernel modules or modification, they can't run on the shared kernel if those are not implemented and virtualized on it. The last disadvantage of the shared kernel is that other operating systems cannot be used on same OS-level virtualization platform if they don't use the same kernel. To solve some of the problems solutions like CoreOS have been developed. CoreOS [21] is an operating system from where all the unnecessary functions are stripped away and only the bare minimum is left to support OS-level virtualization. The whole operating system is also developed especially with OS-level virtualization in mind by having tools which are meant for container management. The CoreOS is not meant to function as an end user operating system.

7. APPLICATION VIRTUALIZATION

Application virtualization is a virtualization technology that closely resembles the OS-level virtualization. Both of these technologies focus on virtualizing the ABI. The difference between these two virtualization technologies is how they implement the virtualization. The OS-level virtualization implements the virtualization by modifying the operating system kernel and changing its own functionality to provide the virtualization support. In the application virtualization the operating system kernel is not modified [122]. Instead a virtualization layer is created on top of the kernel which intercepts system calls the applications try to make [7]. There is also difference in the level of isolation these technologies provide. In the OS-level virtualization the aim is to isolate the containers from each other completely. In the application virtualization the goal is to isolate the applications from the operating system itself. This separation from the operating system allows applications to be used without requiring them to be installed on the operating system. The application virtualization is used to resolve application conflicts, to provide application migration, to isolate untrusted applications, and to support portable application environments. The difference between the application virtualization and the OS-level virtualization is shown in the Figure 29.

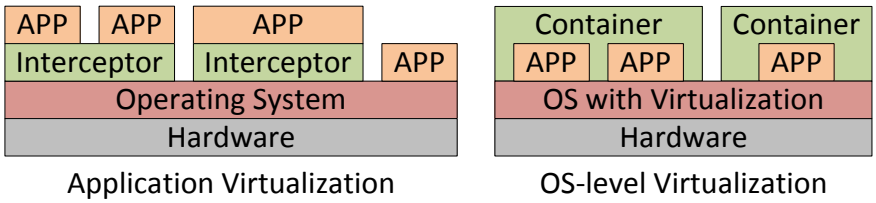

Figure 29. *The difference between application and OS-level virtualization.*

The application virtualization is based on intercepting technologies. The virtualization layer intercepts any system calls applications try to make and then virtualizes them. There are two ways of implementing *the interceptor*: there can be a separate piece of software which works as an interceptor for all the virtualized applications or each application can have its own interceptor. If all the virtualized applications use a common interceptor, this interceptor must be used to launch the application. If each application uses its own interceptor, the application and the interceptor can be combined into one single executable file that can be used to launch the application. The interceptor can be implemented in supervisor mode or in user mode. The benefits of implementing the interceptor in supervisor mode is that the interceptor has better view of the system, it allows to virtualize installers and system processes, and inter process communication

becomes easier. User mode implementation on the other hand offers better protection. [7]

The interceptor is used to capture system calls which modify the operating system's state somehow. What system calls are intercepted depends on the operating system on which the virtualized application runs on. Any system calls that modify files in a file system are examples of system calls which modify the state of the operating system [7][40]. On windows operating systems any system calls which modify the registers of the operating system are also system calls which need to be intercepted [7][122]. When such system call is used from the application, the system call is modified by the interceptor. For example any file writes can be redirected to a single directory or they can be redirected to memory instead [7]. When the same file is read, the read is directed to the single directory or to the memory. This allows any modifications the applications do to be contained. If the writes are directed to memory, the state of the operating system stays unchanged as well as no files are written to the file system and other applications stay unaware of the existence of the virtual application. This doesn't prevent the virtual application from communication with other applications with inter process communication. The downside is that any changes done in the virtual application won't stay in effect unless they are specially written to files. This redirection of modifications to some contained area is also called as *sandboxing* [41].

Using the interceptor for the isolation also allows packaging all the dependencies of the application into a one place. All the pieces of software and data which the application needs are packaged into a single location. For example if the interceptor and the application are combined, all the dependencies can be combined into the same executable. The data and the software dependencies are loaded into memory when the virtual application launches. When the application uses one of its dependencies, the interceptor redirects the calls to the proper memory location. [122][40] The packaging can be done for example by taking two snapshots of the operating system: one before an installation of the application and one after the installation of the software. The required dependencies can be deduced from the changes to the operating system and the package can be created. [122]

The packaging of the whole runtime environment of an application into single file provides many benefits. The most important benefit is the ability to migrate the application easily. When the entire runtime environment of the application is stored in a single file, the application can be moved to other machines and no installation is needed. The virtualization layer makes sure that the applications have the access to the packaged dependencies. [7][122][41] The file can also be divided into small data blocks. These data blocks can then be transferred over a network. This is called as *application streaming* [40]. In the application streaming the virtualized application is launched on some other machine through the network. First some data is transferred over the network to the machine from where the virtual application was launched. This data is stored in memory.

When there is enough data, the application is launched. After this the data is transferred to the machine when the application needs it. The application streaming allows storing the virtual applications in one centralized location. [122] This makes the management of the applications easier. Instead of installing the applications to each machine that needs them, the applications can be made into virtual applications which can be streamed to the machines and no installations are needed. Simultaneously the update management becomes easier as there is only one application which needs to be updated.

The isolation of the application virtualization is limited to the state of the operating system. In application virtualization it is not possible to manage how the hardware resources are shared between the applications. It is the operating system kernel's job. The interceptor implemented in supervisor mode offer less protection than the interceptor implemented in user mode [7]. If there is vulnerability in the interceptor, it can compromise the whole system. In addition if there are any bugs in the code, it might crash the whole system. The user mode interceptors don't have these problems and other applications and the operating system is protected from the virtual applications [122][41].

8. VIRTUALIZATION SECURITY

The virtualization technologies add more layers to the computer systems. As many of the virtualization technologies work at some other privilege mode than user mode, the security of these technologies can affect the security of the whole system. The virtualization technologies can provide some benefits for the security of the systems, but at the same time they also have some risks and challenges which must be addressed. This chapter focuses on explaining the different aspects of the virtualization security.

8.1 Benefits

The greatest benefit of many virtualization technologies is the isolation which they provide [94][97]. All the virtualization technologies focus on sharing resources and decoupling the components from each other. For example, if a hypervisor decouples the guest operating systems from the hardware with emulation and it shares the resources between the emulated hardware properly by using scheduling and QoS functions, an infected guest operating system which does malicious activities won't affect the other guest operating systems running on the same machine [97]. From the virtualization technologies the hardware virtualization can be considered to offer the most isolation because of the small interface that needs to be virtualized. The application virtualization offers the least isolation as only the functions which try to modify the operating system state are virtualized. The processes are still able to communicate with other processes. The OS-level virtualization is the middle ground between these two technologies. It offers better isolation than application virtualization, but because of the complexity of the operating system kernels, it cannot provide as good isolation as the hardware virtualization can [128]. The isolation allows executing multiple software-stacks with different security levels simultaneously on same machine [94].

Hypervisors can also provide security monitoring with VM introspection. A special introspection feature is implemented in the hypervisor which allows monitoring the ISA instructions and the memory which the VMs use. These instructions can be analysed by special security software which can reside inside some other VM than the VM that is being inspected. [36][30] This allows moving the security tools out from the guest operating systems to a separate location [94]. Now if a guest operating system becomes infected, the infection won't compromise the security tools residing in different VM.

The encapsulation properties and the technologies based on them can also be used to improve security. The systems can be made more resistant against attacks by implementing automatic VM restarts from VM snapshots and VM images if a VM crashes

because of an attack or if a VM gets infected [64]. Multiple copies of the same VM can also be started to provide more processing power to prevent the attacks from taking all the resources and from denying the services the VM provides. Live migration can be used to move the VM away from a host computer to where the malicious attack is directed. [94] It is also possible to use the snapshots for forensic purposes. If an infected VM is noticed, a snapshot can be made of it. This stores the contents of the memory of the VM and special forensic tools can be used to inspect the memory for the malicious software [97]. Even if the VM is not infected, the snapshots can be inspected just in case to notice any malicious software that has not been noticed by the run-time security software. As the snapshot is a self-contained file, it can be processed on some other machine than the machine which runs the VMs to prevent the inspection from taking any computing resources from the VMs.

The hypervisors have additional security benefit as well. As they work directly on top of the hardware and the interface they need to emulate is relatively small, the code needed to implement is smaller than the code needed to implement an operating system [97]. For example the Xen hypervisor has less than 150 thousand lines of code [130] while the Linux kernel has over 16 million lines of code [69] which runs in the supervisor mode. The code which runs in the supervisor mode creates the trusted computing codebase for the rest of the software. Hypervisors provide smaller trusted codebase making them easier to secure than operating system kernels.

8.2 Challenges, Risks and Issues

Virtualization software manages the virtual environments and the underlying architecture is shared between these environments in hardware and OS-level virtualization technologies. This virtualization software works as a single point of failure and any security vulnerabilities affecting the virtualization software can cause devastating effects as the vulnerabilities will affect many isolated environments at the same time [97][64]. Virtualization software can be typically attacked in one of two ways: from the software which is in control of the hardware or from the software which runs in a virtualized environment.

In hardware virtualization the type 2 hypervisor inherits any vulnerability from the host operating system kernel. As the host operating system kernel and the hypervisor share the supervisor mode in the type 2 hypervisor's implementation, the hypervisor can be altered from the host operating system kernel [97][64]. This is a security risk. Type 1 hypervisors don't have this problem as the hypervisor is the only software running in supervisor mode. The alternative is attacking the hypervisor from the VMs. If there are coding errors or vulnerabilities in any of the interfaces the hypervisor provides for the VMs to use, the software running in a VM could use these vulnerabilities to gain control of the hypervisor or gain access to other VMs. This is called as *a VM escape* [97]. For example buggy paravirtualized device drivers could allow sharing data between VMs

causing the isolation between the machines to break. The malicious code can easily identify if the operating system it is running on is executing in a virtual environment. The identifying can be done by checking for hypervisor artefacts in processes, file systems, registry or memory, by checking for any instructions which are hypervisor-specific (paravirtualization), or by checking for hardware devices which are only used in virtual environments (generic virtual hardware devices). Any vulnerability in the virtualization software is a serious risk as the virtualization might be used to virtualize software-stacks with different security levels and malicious software from a VM with low security could attack the virtualization software through the vulnerability and compromise a high security VM. [97]

The type 2 hypervisors have other security risks as well. The VMs share the run environment with ordinary applications. Any malicious application could steal the resources the host operating system assigns for the hypervisor to use. This can affect the performance of the VMs. Additionally the host operating system can possibly be used to read any files the hypervisor writes for the VMs and an application running in the host operating system could be used to extract information from these files [64]. Any convenience functions the hypervisor offers could also be used as an attack vector. For example sharing clipboard between the host and the guest operating systems could be used to transfer malicious data or to read data which one is not supposed to read [64].

The OS-level virtualization can have same kind of problems. It is possible to run software with root privileges inside a container. If this hasn't been taken into account when the virtualization support has been implemented in the operating system kernel, it might be possible for the software to escape the container and read data from the other containers or control and manipulate them. An example of this is the earlier mentioned chroot system call which is used to change the file system root folder [91]. With privileged access to the operating system kernel, the same system call can be used to change the root folder to higher level in the directory hierarchy and to access the other containers. The operating system kernel should implement some kind of protection for this kind of problems. As the ABI is often much larger than the ISA, securing OS-level virtualization technology can be more challenging than securing hardware virtualization technology.

It is also important to note that virtualization doesn't remove any vulnerability from the guest operating systems and the guest applications. If the VMs or containers are connected to each other with a network and if one of the VMs or containers gets infected or compromised, all the VMs and containers in the network can become compromised. [97] The virtualization technologies don't make obsolete any security approaches which are applied to non-virtualized environments.

Virtualized environments provide their own challenges for the security. It can become hard to make sure that the software base is current and up to date. The reason for this is

the easy ability to make backups, snapshots, and to live migrate VMs, containers and applications. One can easily have old versions of software if the software is not properly managed and monitored. It is also possible to lose information with the backups and the snapshots [64]. If some software crashes and it is restored from a backup or a snapshot, all the modifications done after taking that snapshot are lost. If the software has been for example updated and no new backup or snapshot has been made after that, the update is lost after the restore. As it is easy to create new VMs and containers in a virtualized environment, VM sprawl is also a problem which can cause security issues. *VM sprawl* is a term which is used to indicate a situation where more and more VM and container images are created unnecessarily which as a result creates a larger attack surface and increases the security maintenance [97][64]. Last security risk with snapshots and backups is the data which they contain. Passwords and sensitive data can be stored into the images and snapshots. [97][64] If some malicious person gets access to such file, he can use introspection tools to inspect the contents of these files. This can be considered as bad as stealing a physical machine.

The monitoring of the software can become hard with the live migration. Identifying VMs can be harder than identifying physical machines and it might be possible that software migrates from one administration domain to another when changing the host machine with live migration. Data leaks can also happen more easily in virtualized environment. The networks can be virtual networks and it can be hard to make sure that everything is properly configured and the networks are properly isolated from each other. In physical environment the networks can easily be done by connecting physical cables to physical devices. In virtual environment the same is done by filling text fields and selecting options. This is more error prone than simply connecting a cable to a device. Some virtual switches might not even include network monitoring tools which can further increase the change of data leaks in virtual networks [97][64].

The memory management in hypervisors can also cause risks. If the hypervisor uses advanced memory management methods with performance impact, it is possible for malicious software inside a VM to affect the performance of all the VMs for which the memory management methods have been enabled. [9] The DMA-capable devices can cause security issues as well. These security issues are not virtualization specific, but they can affect larger software base in virtualized environment as the hardware is shared between multiple VMs and containers. The DMA-capable devices can access the memory without restrictions and this can be used to modify the behaviour of the hypervisor or to read data to which one is not supposed to have access to [124][88]. This is known as *a DMA attack*. One way of doing DMA attack is by creating fake MSIs. There are few ways these can be done [126]. One can for example access the device configuration space and modify it to make the device do malicious activities. Additionally device drivers can have bugs which can cause the devices to modify memory in places where they shouldn’t modify it.

Hardware-assisted CPU virtualization methods can also be used for malicious purposes. It is possible to hide malicious code by setting it run in the place of a hypervisor in root mode. This allows the malicious code to control any non-root software and to collect and to modify data without the software running in non-root mode knowing it. One example of such malware is the *Blue Pill* [96].

Network card interfaces can also cause security issues in virtualized environment. Some NICs can provide Media Access Control (MAC) -address changing function and promiscuous mode. Promiscuous mode is a mode where the NIC reads any data sent to it even if the data is not meant for that NIC. In a virtual environment this could be used to read the virtual network labels or any other data the networks transfer causing a security risk. The MAC-address changing function could be used to change the network configuration which also can cause security issues. With virtual NICs a hypervisor or an operating system kernel can control these functions, but with device passthrough these functions cannot be controlled. Even the storage area networks can be unsecure in virtualized environment. Fibre channel and iSCSI use clear text format meaning that anyone with an access to the network can read the data moving in the network [64].

The network virtualization technologies provide their own challenges for security. Most of the network virtualization technologies don't have any security mechanisms of their own and the security must be implemented in some other ways. Legitimate but malicious tunnel endpoints could also spoof network identifiers [39]. Some of the labelling protocols might not work with existing network equipment because the network equipment doesn't understand them [34].

8.3 Security Approaches and Recommendations

To secure a virtualized environment, the virtualization software shouldn't allow any code modifications to it during its runtime. DMA-capable devices can cause problem in this case as they can write directly to memory and even modify the virtualization software's code. To prevent this there are few things that should be done. The first thing is to thoroughly test any device drivers the virtualization platform uses to remove any bugs from them which could cause the devices to malfunction and to write in wrong location in the memory. The second thing is to make sure that any virtualized software cannot configure or use the devices however they want. In software-based virtualization this means that the virtualization software captures any tries to use or to configure the devices. This is part of the device emulation and shared I/O device management in the virtualization software. In hardware-based virtualization the devices can be passed through, in these cases an IOMMU should be used to protect the memory with its address translations. [126] The IOMMU can be used even in OS-level virtualization. The usage of the IOMMU for security is advised in all cases as it is hardware based solution which removes code from the virtualization software and lowers the change of errors and bugs in the code.

The virtualization software and the host operating system should be updated regularly. This is to apply any bug fixes for bugs which might compromise the security. If the virtualization software requires a host operating system, security tools should be installed in the host operating system as the host operating system can be attacked. Any unused devices should be disconnected from the host machine. Similarly in VMs and containers any unused virtual devices should be disconnected. This reduces the code which can possibly be used in the virtualization software and which might contain bugs. At the same time it also reduces the possible attack surface. In host operating system any unnecessary applications should be removed to reduce buggy code and the possible attack surface. In virtualization software any unused tools should be disabled as well for the same reasons. [97][64]

The proper configuration is important for virtualized environments. For this reason, before using the virtualized environment, one should check that each virtual device is associated with a correct physical device and that the virtual devices are connected to the correct networks. One should also make sure that the networks don't leak data to other networks in virtual switches, in virtual machines or in containers. As the proper configuration is necessary for good security in virtualized environment, any management networks in the system should be secured. This can be done either by having a dedicated physical network for the management communication or the management communication needs to be encrypted with some security protocol. Mandatory access policies need to be implemented as well for the access to the management interface of the virtualization software components. The access for the virtualization software's management interface should be allowed only for administrators of the virtualized environment. If virtualization software uses host operating system for its functionality, the access to the host operating system must also be restricted as it controls the virtualization software. [97][64]

The network virtualization technologies don't often have any security functions. For this reason some special security protocol needs to be used [39]. Typical protocols which are used in virtualized environments are: Internet Protocol Security (IPSec), Secure Sockets Layer (SSL) / Transport Layer Security (TLS), and Secure SHell (SSH) [64]. These protocols are used to encrypt the sent data before it is sent to virtual networks. These protocols allow two endpoints to authenticate each other and agree on things like security keys and tokens before sending the data. The security endpoints should be selected to be in such way that sensitive data doesn't leak into networks which are not secure. Similarly if storage networks and devices are shared between VMs or containers, any sensitive data should be encrypted before sending it to a storage network or storing it to a storage device. This can be done in software or alternatively storage devices can provide encryption services.

Virtual network endpoints should be selected in a way which prevents data from leaking from one virtual network to another. For example the virtual switch in the virtualization

software has to work as the endpoint if the physical host can have VMs or containers that belong to different virtual networks. If all the VMs and containers on same physical host belong to a same virtual network, the endpoint can be the first physical switch to which the physical host is connected to. If a NIC or a virtual NIC (vNIC) provides promiscuous mode, it should be only connected to a network which is assumed to be unsecure. The reasoning for this is that if malicious software gains control of the NIC or vNIC, it could read any sensitive data from the network. If the network is assumed to be unsecure, no sensitive data should be transported in it and the malicious software won't gain any useful information by collecting the data in that network. NICs and vNICs which can change their MAC-addresses should only be connected to networks which are not critical and are unsecure. The reasoning for this is that if malicious software gains control of the NIC or vNIC, it could change the MAC-address to some other MAC-address which is already in use in the network. This can cause the data to be forwarded in unpredictable ways and some of the data will end up for the malicious software.

Monitoring tools should be deployed in the virtualization software. These tools can be special security tools developed for virtualized environments or simply tools which monitor the resource usage. The special security tools can improve the security in some cases compared to non-virtualized case. If guest operating system cannot use any modern security tools, the security tool can use VM introspection to monitor the guest operating system and to provide security that way. If the virtual switches the virtualization software uses don't provide monitoring, the traffic should be routed through software or hardware which can provide the monitoring. [97] This can be for example a separate VM where the monitoring software resides and all the traffic is forwarded through it. A physical switch with monitoring capabilities can also be used.

As the virtualization technologies don't remove any security issues in the software that is virtualized, the VMs and the containers should be treated as if they are their own physical machines. The same security measures should be applied to them as one would apply to the physical counterparts. Especially any VMs and containers which are connected to networks should be secured properly to avoid any malicious software from gaining access to the networks. [97][64] In virtualization environment it is possible to use security tools which are specialized for the virtual environment. A one example of such tool is a driver-based antivirus tool. Instead of having own antivirus agent in each of the VMs, there is only one antivirus agent in its own VM. Special small weight driver is installed in all the VMs which are supposed to be monitored by the antivirus software. [33]

The physical security is also important in the virtualized environments. Each physical machine contains more software than in traditional non-virtualized environment. If anything happens to a physical machine, the consequences are more severe in virtualized environment. [97] Additionally any configurations can be ruined by simply changing

the physical network layout. SDN can be used to reconfigure the virtual network layouts in case the physical network layout is changed. The access to the physical hardware should be controlled and only the administrators should be allowed an access to the hardware.

Any data the software in virtualized environments use is stored into the storage devices. As the data can contain sensitive information, the data should be properly isolated by the virtualization software or by using storage virtualization methods for each VM and container. VM and container images are also stored in storage devices along with any snapshots which have been taken from the VMs or containers. These files can contain sensitive information as well which means that the access to them should be managed. The access to the storage devices should be controlled by mandatory access policies and with physical security. Any data should be backed up often and into different storage device to avoid problems caused by device malfunctions. VM and container images should have backup copies made of them as well. Snapshots should be done periodically to minimize any data losses caused by crashes or malfunctions. In order to increase the security of the stored images and snapshots, cryptographic checksums should be calculated for them periodically. The checksums can be then used to identify if any modifications have been done to the images or snapshots. Different kinds of security tools should also be used to scan the contents of images and snapshots periodically to detect rootkits, viruses or malware. [97]

9. VIRTUALIZATION IN INDUSTRIAL AUTOMATION

Most of today's virtualization technologies are relatively new and haven't been used for that long even in traditional IT environments. As a result of this there is relatively little publicly available research that has been done of using virtualization in industrial automation systems. In this chapter some of the previous research and literature that has been done of this subject is presented, some virtualization solutions are compared, the virtualization applicability to industrial automation is analyzed, and a suggestion is made how virtualized industrial automation system could be implemented. Lastly some suggestions for future research work are presented.

9.1 Previous Research and Literature

H. P. Breivold and K. Sandström researched usability of virtualization for testing purposes in industrial automation in their paper [12]. First they present some hardware-related challenges in software testing (availability of test infrastructure, fault tolerant facilities, flexibility of test environment setup, and fault reproduction for debugging and cost) after which they give a very short overview of virtualization technologies and present some technologies addressing the challenges. Lastly they analyze some of the impacts virtualization has for industrial automation requirements. The virtualization helps with allowing multiple instances of software to be tested on same hardware, changing the testing environment configuration easily and allowing functionality like injecting faults to the virtual machines the software is running on. In addition the save and replay functionality is suggested for debugging. The conclusion of the paper is that the virtualization can be used to assist in testing the functionality of software, but it cannot replace the testing on proper hardware as the virtualized environment has different performance, timing, determinism, reliability, availability and safety properties than typical industrial automation environment.

Wenbin Dai et al. presented in their paper a configurable cloud-based testing infrastructure for interoperable distributed automation systems [23]. The paper describes a platform which provides testing capabilities for software. The platform provides virtualized devices which can be used to test the software. The virtualized devices emulate the behavior of the actual physical devices. This is simple software I/O virtualization. The I/O calls directed to the virtual devices can be handled in two different ways. The first method is to return pre-recorded data for the virtual device which then emulates the behavior of the devices at a correct moment for the software using it. The second method

is to retrieve the necessary data from some external source in real-time. This could be for example some kind of process simulation or an actual process.

There are some research and literature done of the security of virtualized industrial automation systems. Linking the Oil and Gas Industry to Improve Cybersecurity (LOGIIC) is a program which aim is to research and develop the level of cybersecurity in critical industrial automation systems. Part of this program is researching the security of virtualized industrial automation systems and one document has been made publicly available of this subject [74]. This document describes how existing virtualized automation systems were tested and what kind of results was achieved. The testing was focused on the 2-3.5 levels of the automation system. This means that the levels 1 and 4 are not focused on the research. The testing done for existing virtualized industrial automation systems consisted of three steps: reconnaissance, information capture / data retrieval attempts and targeted attacks. The architecture and the systems tested were considered to be known when testing the security issues of the systems. Test vector consisted of things like hardware, shared memory, software and hosts, security of the VM OSes, VMware and Hyper-V hypervisors, functionality testing, patch management, and update mechanisms. The testing was done by using tools like Kali Linux distribution for penetration testing, Wireshark, computer forensic tools, custom attack scripts, existing exploits, and reverse engineering tools. The detailed results and methods are not presented in the document. The most important finding according to the document is that the security of a virtualized industrial automation system needs to be considered during the whole lifecycle of the system as the virtualization technologies change how the systems are built. The hypervisors and their management software is part of the security considerations. The findings of the research also indicate that the proper configuration of the system is important. Networks need to be configured properly to avoid cross communication between networks and perimeter security is required. Patching was found to be really important for the hypervisor as many of the vulnerabilities were found because patches were not applied to the hypervisors. The research was fully focused on empirical research of security vulnerabilities on existing virtualized industrial automation systems which use either VMware or Hyper-V hypervisors and the detailed description of the research is not publicly available.

Another security related document is one of the Intel's white papers in which they discuss how an open source hypervisors along with Intel's special security hardware can be used to securely launch hypervisors and guest operating systems [134]. The paper is focused on industrial automation systems. Intel's *Trusted Execution Technology* (TXT) was used to provide security for the system. The TXT is a hardware-based component which verifies launched software to be trusted. It does this by comparing critical elements of the software against a good known source. It uses cryptographically generated digital signatures for this and the digital signatures are generated with a *Trusted Platform Module* (TPM). The TPM is a hardware component which purpose is to generate

cryptographic keys. The TXT provides security only at software launch. Other security solutions are required after that. The paper uses Xen as their hypervisor and a pre-kernel module called *Trusted Boot* is used to enable the Xen to use the TXT. Both the Xen and the Trusted Boot are open source software. With the TXT the hardware prevents launching software which has been compromised by some malicious software.

The Rockwell Automation has made a document [119] describing guidelines for virtualizing industrial automation systems by using VMware's hypervisor and their automation software. The document gives guidelines how to select hardware and assign resources for the software. The contents of the document cannot be generalized for other industrial automation systems because the document is made for specific software made by the Rockwell Automation.

There are many different papers discussing about real-time virtualization and providing hard real-time guarantees with virtualization platforms. N. Mahmud, K. Sandström and A. Vulgarakis research in their paper if it is possible to use virtualization to virtualize and run multiple industrial automation controllers on same hardware [72]. They use Xen as their virtualization platform and they use real-time Linux (Yocto Linux Distribution) for their guest operating systems. Virtual switches (Open vSwitch) are used on physical machines to direct the traffic. The connections used between physical machines use normal Ethernet without any real-time capabilities. Two testing set ups were made: one without virtualization where two real time Linux operating systems run on separate computers and communicate through an Ethernet connection and one with virtualization where two physical machines both host four virtual machines and the communication is through a shared Ethernet connection. In the virtualized case the computers support hardware-assisted CPU and memory virtualization and IOMMU. All the virtual machines were pinned to their own CPUs to avoid problems with vCPU scheduling. Three different things are tested: latency and jitter, network throughput, and CPU load. The latency and jitter testing showed that the average latency is around 90 microseconds worse than in the case without virtualization and the average jitter was around 11 microseconds worse. However, the average latency was less than 400 microseconds and the average jitter was less than 20 microseconds for a round-trip time meaning that the virtualization could be used for most real-time applications. In network throughput no significant difference was noticed. It was noted that computation, which took 10.16 seconds on a non-virtualized machine, took 12.13 on virtualized machine.

Intel has also published a paper where real-time capabilities of virtualization are tested [2]. They test how much latency and jitter the system has between sending an interrupt from a PCI device and receiving an answer from a real-time operating system. Hypervisor used was Wind River's hypervisor and the real-time operating systems were Wind River's VxWorks. General purpose Wind River Linux operating system was running alongside the real-time operating systems. The average interrupt latency they got in virtualized case was 5.89 microseconds while the minimum latency and maximum latency

were 4.90 and 12.38 microseconds. The paper concluded that this latency would be suitable for control loops in the 100-300 microsecond range and that their test was a best case scenario.

S. Ghosh and P. Sampath have also tested real-time virtualization for industrial control in their paper [38]. In their experiment they used VT-x enabled hardware. The hypervisor used was type 2 hypervisor *Kernel-based Virtual Machine* (KVM) and the host operating system was Ubuntu with rt-preempt real-time patch. The machine hosted three virtual machines: two hosting general purpose operating systems and one hosting real-time controller. The output of the system was monitored by an oscilloscope and the communication protocol used was EtherCAT. The test was conducted with different levels of stress applied to the system. With the test it was noticed that the real-time controller was able to provide average scan time of 592 microseconds with jitter being around 30-40 microseconds. This is suitable for most industrial applications as well. It was also noticed that the extra stress added to the system had negligible effects to the real-time execution of the controller.

There has also been some research about virtualizing *Controller Area Networks* (CANs). Controller area networks are commonly used in automation systems to provide communication between different components in the system. H. Herber et al. describe in their paper two different implementations for virtual CAN controller which can be used with hypervisors [45]. They implement paravirtualized CAN controller and in addition they implement SR-IOV enabled CAN controller. In their tests the paravirtualized CAN controller added around 35 microseconds extra latency to a complete transmit-receive loop compared to a non-virtualized case. The SR-IOV enabled controller added around 20-25 microseconds extra latency to a complete transmit-receive loop compared to a non-virtualized case. L. Niemistö implements virtual CAN bus in his thesis [83]. The virtual bus consists of two elements: a virtual device driver and a virtual CAN hub. The virtual CAN hub implements multiple channels on top of a physical CAN bus and the virtual device driver is used to access the virtual CAN hub.

9.2 Benefits and Challenges of Virtualization

The virtualization can offer many beneficial features for industrial automation systems. The virtualization can be used in two ways to benefit the industrial automation systems: it can be used for testing and development or it can be used in live systems.

9.2.1 Testing and Development

The testing in industrial automation can be costly because of the complexity and the requirements of the automation systems. Virtualization can be used to assist with the testing and to lower the development costs. Instead of every developer needing their own hardware to test software, shared hardware can be used and each developer can be

provided a virtual environment where to test the software. This saves in hardware costs. Changing the test environment can also be done more easily in a virtual environment than in a physical environment. In a virtual environment new virtual machines can be created easily and virtual networks formed between them. The configurations of the machines can also be saved into snapshots and images. In a traditional environment one would have to buy new machines, connect them physically and then manually configure each new machine.

The virtualization can also be used to inject errors and faults to the systems being tested. For example I/O calls which are sent to device emulators or back-end drivers can be routed to a backend testing tool instead of the native device drivers. This testing tool can then be set to generate outputs which emulate hardware faults for the emulated or paravirtualized devices. The record and replay functions of hypervisors can also be used for testing purposes. Many of the real-time software in industrial automation systems work on single CPU because it's simpler to develop and it provides better determinism. However, some non-real-time software might work on multiple cores. For example data collection services could work on multiple cores to improve the performance. It is known that debugging concurrent programs can be hard as the execution is not necessarily deterministic and different parts of the software can execute in different order every time the software is executed. This makes it hard to reproduce some of the bugs in the code. Record feature of the hypervisors can be used to record the execution of the software when testing it and if a bug is encountered, its execution can then be replayed and the bug analyzed from the recorded execution. The bug can also be easily repeated in this manner. It is important to note that if virtualization is used to test software which is not deployed in a virtual environment, the actual execution of the software differs from what it will be in the actual run-time environment. The reason for this is the inherent overhead the virtualization technologies have from intercepting different calls and instructions and labeling different entities to recognize them and to isolate them from each other. In a virtual environment the resources are also shared. This is why the software should be tested in an actual physical environment as well.

9.2.2 Live Systems

In live systems the virtualization can provide many benefits as well. The most obvious benefit is the reduced amount of hardware which is needed. Traditionally pieces of software are isolated from each other by having them executed on separate physical machines. With virtualization the isolation for the pieces of software can be provided within a single machine. As a result of this the costs of building an industrial automation system can be lowered. The smaller amount of hardware also lowers the operational costs of the industrial automation systems. Because there is less hardware being used, less electricity is required to run the hardware and the energy costs will be lower than in a non-virtualized environment. At the same time the hardware becomes better utilized

as the computing resources can be used more efficiently between pieces of software. With hardware virtualization there is no need to keep old machines to host old operating systems and software to configure old equipment. If the software is needed, a new VM can be created with the old operating system and the software so that the equipment can be configured.

The virtualization can also ease administration and decrease the required deployment time for industrial automation systems. The deployment time can be decreased as the required environments for pieces of software can be preconfigured and packaged into images and snapshots. The images and snapshots can then easily be deployed to the industrial automation systems. Similarly any patches can be deployed with images. A patched version of software is packaged into an image along with its environment and then the new version of the software can be launched into the virtual environment. As the whole environment of the software is packaged into same image, the patched software can be fully tested before its deployment into the live industrial automation system. In addition the images can be used to provide no downtime during the software patching operation. When a patch is required to be applied, a new virtual machine or container can be started with the patched version of the software. Once the patched version of the software is up and running, any traffic can be redirected to it from the old unpatched software version. If the hardware is required to be updated or faulty piece of hardware must be replaced, live migration can be used to move software to other physical machines for the duration of the maintenance.

Many of the features which virtualization technologies offer can be used for fault tolerance and disaster recovery. The ability to create new virtual machines and containers quickly can be used as disaster recovery method. If some software crashes or hardware becomes faulty, a new VM or container can be started somewhere else and the traffic can be redirected to that VM or container. Similarly live migration can be used if a faulty device is noticed on the physical machine to transfer the VMs and containers to a physical machine with working hardware.

Applying virtualization to industrial automation systems has its challenges. All the virtualization technologies add some form of overhead. For example in hardware virtualization the ISA instructions that would change the state of the hardware need to be trapped. Similarly in OS-level and application virtualization any system calls which would modify the state of the operating system need to be intercepted. In network and storage virtualization extra overhead comes from adding labels to the messages and doing address translations. The problem with many address translation mechanisms is also that they will cause jitter when used. For example the address translation in MMU and IOMMU use TLBs to store translated addresses and to speed up the memory access. The problem is that the size of the TLBs is limited and old translated addresses are removed in order determined by some algorithm. When there are multiple different VMs executing on same machine, the VMs can execute in some order determined by the hy-

pervisor. Any software in the VMs can access the memory and cause some translated address in the TLB to be replaced. Now the replaced address must be translated again when the memory in that address is accessed. This will cause there to be different durations for the memory access operations causing some latency variation. In addition the different level caches can cause variation for the memory access. If the shared cache is not isolated between VMs, one VM could replace all the data in the shared cache and other VMs would have to retrieve the data they need again from the memory. This causes jitter as well for the memory access as retrieving data from memory is slower than retrieving it from the shared cache.

The resource sharing can also be a problem with virtualization. It is possible to overcommit the resources so that all the pieces of software which require the resource are not able to use them. The underlying physical hardware needs to have enough resources available to support all the VMs and containers on that hardware. The resource sharing algorithms might not be suitable for industrial automation systems either. Many of the virtualization solutions are made for traditional IT environments where real-time applications are not that common. As such the resource schedulers are not designed to provide real-time capabilities.

In some rare cases licensing can cause problems as well. Some licenses are tied to hardware and in a virtual environment the hardware is virtual hardware. If the license is tied to the virtual hardware, it can cause problems when the VM is live migrated and the composition of the virtual hardware might change.

The specialized devices in industrial automation are also a problem. Each device that is used in a virtualized environment requires it to be virtualized if it is used with multiple VMs or containers. However, the devices used in industrial automation systems are not often used in traditional IT environments and as such the virtualization solutions don’t include device emulation or paravirtualized device drivers for such devices. The devices used in industrial automation systems don’t also include hardware-assisted virtualization support because virtualization hasn’t been used in these systems.

The last problem is the complexity of virtual environments. Using and configuring virtualization tools requires a specialized skill set. It is easier to configure the system badly in a virtual environment than it is in a physical environment. Industrial automation systems are often managed by plant technicians and process engineers without proper training or understanding for virtual environments. This can cause security and safety risks if the virtualized environment is managed by a person without the required knowledge.

9.3 Comparison of Different Virtualization Solutions

In order to estimate the applicability of virtualization to industrial automation, some virtualization solutions’ features are compared. The virtualization solutions for the

comparison were selected based on the popularity of the solutions, the suitability for real-time applications, and the availability of the technical information. From hardware virtualization solutions the following hypervisors are compared: Hyper-V, KVM, VMware ESXi (vSphere 6.0), VMware Workstation 12 Pro, Wind River Open Virtualization, Xen 4.6 and XtratuM. From OS-level virtualization solutions the following virtualization solutions are compared: Docker, Linux-VServer, LXC, LXD, OpenVZ and Virtuozzo. From application virtualization solutions the following solutions are compared: BoxedApp, Cameyo, CDE, Microsoft App-V and VMware ThinApp.

9.3.1 Hardware virtualization

Seven hypervisors were selected to be compared. Three of the hypervisors are proprietary and the rest of the hypervisors are open source projects. The proprietary hypervisors are the Microsoft's Hyper-V hypervisor [113] and the VMware's ESXi and Workstation 12 Pro hypervisors [120]. The open source hypervisors are KVM [58], Wind River Open Virtualization [125], Xen [130] and XtratuM [132]. From these the Wind River Open Virtualization is not really its own hypervisor, but instead it uses the KVM and other open source projects to create a modified hypervisor for real-time purposes. All the hypervisors take slightly different approach for the hardware virtualization. The general information of the hypervisors is collected into the Table 1.

Table 1. *The general information of the compared hypervisors.*

Hypervisor	Type	Supported host CPU Architectures	Host OS	License
Hyper-V	1	x86-64	Windows Server 2008 or Windows Server 2012	Proprietary
KVM	2	x86-64, ARMv7, ARMv8, PowerPC, S390	Linux, FreeBSD	Various
VMware ESXi	1	x86-64	Doesn't use one	Proprietary
VMware Workstation 12 Pro	2	x86-64	Windows, Linux	Proprietary
Wind River Open Virtualization	2	x86-64, ARMv7, ARMv8, PowerPC, S390	Linux, FreeBSD	Various
Xen	1	x86-64, ARMv7, ARMv8	Linux, FreeBSD, Hurd, NetBSD, OpenSolaris	GNU GPL2
XtratuM	1	x86, ARM, LEON2, LEON3, LEON4	Linux, LithOS, µLithOS, Partikle, RTEMS, Android, Meego	GNU GPL2

The x86-64 CPU architecture support on the table means that the hypervisors support also the 64bit machines on the x86 architecture. In the table there are host operating systems marked even for the type 1 hypervisors which execute alone in the supervisor mode. The reason for this is that these hypervisors use privileged VMs for the device

drivers and management software. The marked operating systems are operating systems which can be used in the privileged VMs for the hypervisor. In the license column the "various" indicates that the project uses various different open source licenses.

One of the responsibilities of a hypervisor is to emulate and to share the physical CPUs. Different hypervisors have different methods for doing this. The features meaningful for industrial automation virtualization are shown in the Table 2.

Table 2. *The CPU virtualization features of the compared hypervisors.*

Hypervisor	Supported virtualization methods	Schedulers	Co-scheduling	Real-time support
Hyper-V	Hardware-assisted virtualization	Proprietary	No	No
KVM	Hardware-assisted virtualization	Depends on the host OS	Depends	Depends
		FIFO	No	Yes
		Round robin	No	Yes
		CFS	No	No
VMware ESXi	Full virtualization Hardware-assisted virtualization	Relaxed co-scheduler	Yes	No
VMware Workstation 12 Pro	Full virtualization Hardware-assisted virtualization	Depends on the host OS	Depends	Depends
Wind River Open Virtualization	Hardware-assisted virtualization	Depends on the host OS	Depends	Depends
Xen	Paravirtualization Hardware-assisted virtualization	Credit	No	No
		Credit 2	No	No
		RTDS	No	Yes
		ARINC 653	No	Yes
XtratuM	Paravirtualization	ARINC 653	No	Yes

Many of the virtualization solutions support CPU virtualization only with the hardware-assisted virtualization technologies. Only the VMware's products offer full virtualization methods. The VMware used to offer paravirtualization as well, but the support has been removed from the newer products when the hardware-assisted virtualization technologies became more common [120]. Only the Xen and the XtratuM support paravirtualization.

The Hyper-V uses its own proprietary CPU scheduler. Any vCPU that wants to execute informs the scheduler about it. The scheduler selects vCPUs to run on free pCPUs in a way which allows each vCPU equal change to execute. This makes the scheduler not suitable for real-time applications where the execution needs to be deterministic. The VMs can be set CPU usage reservation and usage limits to control the CPU usage. No co-scheduling is supported as the guest operating systems are expected to be designed to

run in virtual environments where vCPU skew can exist. The CPU scheduling can be made NUMA-aware. [90]

The KVM uses by default the Linux's default *Completely Fair Scheduler* (CFS) which tries to schedule the VMs fairly [58]. It does not support real-time scheduling or co-scheduling. Other possible schedulers in Linux are the *First In First Out* (FIFO) and *Round Robin* schedulers. With these schedulers it is possible to set priorities to different VMs which can be used to prioritize soft real-time VMs. Additionally the round robin scheduler allows limiting the time one VM can use for its operation. Co-scheduling is not supported as these are the operating system kernel's schedulers and not schedulers made for a vmkernel. Linux is NUMA-aware so KVM becomes naturally NUMA-aware. [28] However, all these properties depend on the actual host operating system the KVM uses. One can for example apply a real-time patch (rt-preempt for example) to the Linux allowing the KVM to use the hard real-time schedulers provided by that patch.

The VMware ESXi uses a special made vmkernel. Special relaxed co-scheduler has been made for this kernel. It tries to schedule different VMs fairly while taking into account the vCPU skew. The scheduler counts the time vCPUs execute and tries to keep the vCPU skew within some limits for a VM. The scheduler doesn't support real-time VMs. The scheduler is NUMA-aware. [120]

On the VMware Workstation 12 Pro the scheduling is dependent of the scheduler used by the host operating system. As such the real-time properties depend on the operating system used. Co-scheduling is most likely not used as operating systems don't need such feature.

The Wind River Open Virtualization is based on the KVM. As such it is dependent of the scheduler in the host operating system as well. Yocto Project is used to create operating systems for embedded environment in the Wind River Open Virtualization. [125] These operating systems can use the rt-preempt patch. Because of this, the Wind River Open Virtualization solution provides better real-time capabilities than a general purpose Linux with the KVM.

The Xen hypervisors supports currently only the credit scheduler. This scheduler is a proportional fair share scheduler. This means that the vCPUs are scheduled based on the total CPU usage and weights set to the vCPUs. Co-scheduling and real-time are not supported with this scheduler. The credit 2 scheduler is improved version of the credit scheduler and it is currently experimental. The Real-Time-Deferrable-Server (RTDS) scheduler is a real-time scheduler which follows preemptive global earliest deadline first scheduling policy. It's currently experimental and doesn't support co-scheduling. Lastly the ARINC 653 scheduler is specially made for real-time scheduling. It is based on the ARINC 653 specification which is made for aviation domain to specify domain

isolation on same hardware. The scheduler schedules predetermined set of VMs periodically repeatedly with fixed periods. As such the scheduling is perfectly deterministic. Currently the scheduler doesn't support multi-core scheduling and the support for the scheduler is not clear. The lack of multi-core support means that each pCPU has to have its own scheduler. The Xen supports NUMA-aware scheduling. [130]

Lastly the XtratuM uses only the ARINC 653 scheduler for its vCPU scheduling [132]. Scheduling for multiple cores is not possible as the ARINC 653 specification doesn't specify it. As such the scheduler cannot be NUMA-aware and co-scheduling is not supported.

Supported memory virtualization methods for the hypervisors are shown in the Table 3. Almost all the hypervisors use the same memory virtualization methods.

Table 3. *The memory virtualization methods of the compared hypervisors.*

Hypervisor	Supported virtualization methods
Hyper-V	Full virtualization Hardware-assisted virtualization
KVM	Full virtualization Hardware-assisted virtualization
VMware ESXi	Full virtualization Hardware-assisted virtualization
VMware Workstation 12 Pro	Full virtualization Hardware-assisted virtualization
Wind River Open Virtualization	Full virtualization Hardware-assisted virtualization
Xen	Full virtualization Paravirtualization Hardware-assisted virtualization
XtratuM	Paravirtualization

All the virtualization solutions apart from the XtratuM support both full and hardware-assisted memory virtualization. The reason why the XtratuM lacks the hardware-assisted virtualization is because it is more focused of virtualizing LEON processors which lack the hardware-assisted virtualization support. Instead the XtratuM uses paravirtualization for the CPU architectures which have a MMU for its better performance compared to full virtualization. [132] The Xen hypervisor is the only other virtualization solution which offers paravirtualization for the MMU virtualization [130]. Other important feature when considering using virtualization for industrial automation is the Intel's Cache Allocation Technology (CAT) as it prevents VMs from affecting each other with the shared memory cache. However, the CAT is so new technology that most of the virtualization solutions don't have support for it yet. Only the Xen hypervisor has official support for it currently [130].

In I/O virtualization the hypervisors need to provide support for different virtualization methods. Some of I/O virtualization features are shown in the Table 4.

Table 4. *The I/O virtualization features of the compared hypervisors.*

Hypervisor	Supported virtualization methods	Virtual switch	File system	Volume manager
Hyper-V	Full virtualization Paravirtualization Partial hardware-assisted virtualization	Yes	Yes	Yes
KVM	All	Yes	Yes	Yes
VMware ESXi	All	Yes	Yes	Yes
VMware Workstation 12 Pro	Full virtualization Paravirtualization	Yes	Yes	Yes
Wind River Open Virtualization	All	Yes	Yes	Yes
Xen	All	Yes	Yes	Yes
XtratuM	Paravirtualization Hardware-assisted virtualization	No	No	No

Most of the virtualization solutions provide support for all of the virtualization methods. The Hyper-V provides full virtualization and paravirtualization methods for devices. Hardware-assisted virtualization is not as well supported. Only some devices which have proper drivers can support the device passthrough. For example storage devices can be passed through. [113] The VMware Workstation doesn't support device passthrough because the host operating system is the one in control of the device drivers and the hypervisor doesn't have control of the host operating system. The XtratuM doesn't provide full virtualization as the device emulation would complicate the code and it would provide worse performance than paravirtualized devices.

All of the hypervisors apart from the XtratuM provide some kind of virtual switch, file system and volume manager. These are not provided by the XtratuM as the goal has been to make its code base small as possible and easily certifiable [132]. The virtual switch provided by the Hyper-V provides basic layer 2 functions and some more advanced functionality like network traffic monitoring and bandwidth limiting. In addition the functionality of the switch can be expanded with extensions which can for example be used to enable SDN. [113] The KVM uses a virtual switch provided by the QEMU emulator the KVM uses for device virtualization [58]. The full virtualized or paravirtualized NICs are connected to this switch and the switch can be connected to the host machine's network. The switch doesn't offer SDN. If SDN is desired, one can use open source virtual switch called Open vSwitch as it supports OpenFlow protocol. The VMware ESXi provides a basic virtual switch [120]. This switch doesn't provide QoS functions. QoS and SDN capable virtual switch can be purchased as a separate software

component in a virtualization product called VMware NSX. Similarly the VMware Workstation provides a basic virtual switch, but more advanced switch can be bought with the VMware NSX. [120] The Wind River Open Virtualization hypervisor uses the Open vSwitch in the KVM hypervisor instead of the QEMU switch to provide SDN capabilities. [125] The Xen provides two possibilities as well. A basic virtual switch can be used which provides the basic layer 2 functionality or the Open vSwitch can be used to take advantage of more advanced features. Both of these provide basic QoS functions for rate limiting. [130] The Open vSwitch also supports VXLAN, GRE, STT, GENEVE and IPsec protocols.

The hypervisors which use a host OS typically use the file system and the volume manager provided by that host OS. The VMware ESXi implements its own filesystem and volume management [120]. For example the KVM and the Xen use the virtual file system and the logical volume manager provided by the Linux operating system.

Lastly some features which can be useful in virtualized industrial automation systems are compared. These features are live migration, save and restore, virtual trusted platform module, virtual machine introspection, fault tolerance, and tracing. These are shown in the Table 5.

Table 5. *The extra features of the compared hypervisors.*

Hypervisor	Save / restore	Live migration	Virtual TPM	VM introspection	Fault tolerance	Tracing
Hyper-V	Yes	Yes	No	No	Yes	Yes
KVM	Yes	Yes	No	Yes	No	Yes
VMware ESXi	Yes	Yes	No	Yes	Yes	Yes
VMware Workstation 12 Pro	Yes	No	No	No	No	Yes
Wind River Open Virtualization	Yes	Yes	No	Yes	No	Yes
Xen	Yes	Yes	Yes	Yes	Yes	Yes
XtratuM	No	No	No	No	No	Yes

The live migration is the ability to move a running VM from one physical machine to another. All hypervisors apart from the XtratuM and the VMware Workstation support live migration. The save and restore feature is the ability create VM images and snapshots. All hypervisors apart from the XtratuM support this. The trusted platform module is a hardware security component used for generating digital signatures. Many of the virtualization solutions can use it for their own functionality, but only the Xen supports virtualizing the TPM [130]. As such only the guests on the Xen hypervisor can use the TPM for their own functionality. VM introspection tools allow inspecting the state of the VMs. This can be used to implement isolated security tools outside the VMs. All

hypervisors apart from the Hyper-V, the VMware Workstation and the XtratuM support some form of VM introspection. The Xen and VMware ESXi support it naturally [130][120] while the KVM and the Wind River Open Virtualization require a special tool for it (LibVMI). The fault tolerance is a feature where a redundant copy of a VM is running simultaneously with the primary copy of the VM. If something happens to the primary VM, a redundant copy is made as the primary VM and the execution continues normally with a minimal downtime. The actual implementation of the fault tolerance may vary. For example the Xen takes often snapshots which are copied and merged to the redundant copy to update the copy [130]. The VMware ESXi simply stores the non-deterministic events and injects them to the redundant copy that is running at the same time. Any messages out from the redundant copy are blocked. [120] The last feature is tracing. Tracing is simply a method of collecting data from the hypervisor. This data can be used for example for debugging or finding out things like pCPU usage during the run time [130]. The tracing can also be used for the save and replay functionality. Only the VMware products support the save and replay functionality [120].

9.3.2 OS-level Virtualization

Six different OS-level virtualization solutions are compared. Five of them are open source projects and they all virtualize the Linux operating system kernel. One of the compared solutions is proprietary and it can be used to virtualize both the Linux and the Windows operating system kernels. The general information of the virtualization solutions is shown in the Table 6.

Table 6. *The general information of the OS-level virtualization solutions.*

Virtualization Solution	Base operating system	Virtualization level	License
Docker	Linux	Application	Various
Linux-VServer	Linux	Operating system	GNU GPL2
LXC	Linux	Operating system	Various
LXD	Linux	Operating system	Apache 2 license
OpenVZ	Linux	Operating system	Various
Virtuozzo	Linux Windows	Operating system	Proprietary

In the table the column labeled as “Virtualization level” indicates the granularity of the software packaged into containers. The Linux-VServer, the LXC, the LXD, the OpenVZ and the Virtuozzo all work with operating system level containers. This means that each container has its own set of system libraries and each container hosts multiple applications. The Docker is a bit different and it works with application level containers [27]. This means that one container can only host a one application and if the applications need to communicate with each other, the containers have to be linked together. In the Docker the container images are layered and each layer can be shared between mul-

tiple images. Only the top layer can be written to and all the other layers are read-only. [27] The Virtuozzo is the only OS-level virtualization solution which offers virtualization for the Windows kernel. As the Windows is a proprietary operating system, the virtualization is implemented by implementing a virtualization layer on top of the OS kernel. This layer implements its own proprietary filesystem and service abstractions which are used by the software in the containers. [112] The Linux implementation of the Virtuozzo is based on the OpenVZ [85]. Most of the Linux OS-level virtualization solutions use the existing namespace and cgroup features in the Linux kernel to provide virtualization. The LXD uses the LXC as a base and implements some alternative management tools and extra features to the LXC [68].

The Table 7 shows the different features provided by the different OS-level virtualization solutions. The term limits is used to indicate any methods that can be used to isolate the resource usage between containers.

Table 7. *The features offered by the compared OS-level virtualization solutions.*

Virtualization solution	Save / restore	Live migration	Real-time support	CPU limits	Memory limits	Network limits	Storage limits
Docker	Yes	No	No	Yes	Yes	No	Partial
Linux-VServer	No	No	No	Partial	Yes	No	Partial
LXC	Yes	No	Yes	Yes	Yes	Yes	Partial
LXD	Yes	Yes	Yes	Yes	Yes	Yes	Partial
OpenVZ	Yes	Yes	No	Yes	Yes	Yes	Yes
Virtuozzo	Yes	Yes	No	Yes	Yes	Yes	Yes

The save and restore feature is similar to the hypervisors corresponding feature. It allows storing container images and snapshots and then restoring them. Only the Linux-VServer doesn't support this feature [70]. The live migration is also similar feature to what hypervisors have. It allows transferring containers from one physical host to another. Only the LXD, the OpenVZ and the Virtuozzo have this feature natively. Although not mentioned in the table, a fault tolerance feature is provided only by the Docker. The Docker allows creating swarms where there exists a primary container and multiple secondary containers [27]. If the primary container crashes, one of the secondary containers can take over.

In Linux the control groups (cgroups) can be used to control the access to resources. Each container has its own cgroup. The Linux contains two different schedulers which can be used to schedule the control groups. These schedulers are Completely Fair Scheduler (CFS) and Real-Time (RT) scheduler. [84] While many of the virtualization solutions use cgroups, most of them don't offer interface to change the scheduler. Only the LXC offers interface which can be used to use the RT scheduler [68]. This same interface is also included in the LXD so it can use the RT scheduler as well. Rest of the

virtualization solutions either use the CFS or adapt their own scheduler which works in similar manner allowing fair sharing of the CPUs. It is important to note that the RT scheduler in the Linux only supports soft real-time for containers.

The different OS-level virtualization solutions offer different levels of limitations one can set for the hardware resources the containers use. The Docker can limit the CPU usage of the containers by setting share constraints to each container. A share constraint tells how much of the CPUs the container can use compared to the other containers on the same machine. In addition one can limit what CPUs the containers can use and how long the CFS scheduling period is. Lastly CPU quotas can be set for containers. These limit how much of the CPUs the containers can use at most. [27] On the Linux-VServer one can set hard limits for CPU usage. These limit how much CPU time containers can use at once. [70] The LXC allows using the cgroups supported by the Linux kernel. This allows the LXC and the LXD to use same CPU limits as the Docker uses. For real-time cgroup one can set a period and a maximum runtime. [68] The OpenVZ limits the used CPU with shares and with upper limits for used CPU time [85]. The Virtuozzo provides same kind of limits as the Docker does [112].

All the virtualization solutions can limit the maximum memory the containers can use. In addition many of them can also limit swap file size for their virtual memory. For network limiting the LXC and the LXD offers tagging the network packets with a class identifier which can be used in the Linux Traffic Controller to identify the packets from different containers [68]. The OpenVZ provides a fair I/O scheduler where one can assign priority to containers and the bandwidth is distributed according to these priorities [85]. The Virtuozzo allows setting maximum and minimum bandwidth rates for containers [112].

In the Docker it is possible to limit the bandwidth usage of block I/O. However, the limiting is only done by giving a weight for each container and this weight is used to control how much of the block I/O device each container can use. The used storage space cannot be limited. [27] In the Linux-VServer it is possible to limit the used storage space, but limiting the used I/O bandwidth is not possible [70]. The LXC and the LXD both can use weights similar to what the Docker uses. In addition maximum allowed I/O operations can be specified. The allowed storage space for each container cannot be specified. [68] If the storage space is wanted to be limited, one can use the logical volume manager to create logical volumes for each container. The OpenVZ can limit the storage space the containers use [85]. A fair I/O scheduler is used for the I/O operations. The Virtuozzo can limit the storage space the containers can use, prioritize the I/O operations, and limit the I/O operations the containers can make [112].

9.3.3 Application Virtualization

Five different application virtualization solutions were compared. The BoxedApp [11], the Cameyo [15], the Microsoft App-V [6] and the VMware ThinApp [120] are proprietary solutions while the CDE [40] is open source. The Table 8 shows features of these virtualization solutions. Execution mode indicates if the virtualization software runs in user mode or if it runs in supervisor mode. With these virtualization solutions it also indicates if each application has its own virtualization software or if the virtualization software is common for each virtualized application. Solutions working in user mode provide virtualization software for each virtualized application while the solutions working in supervisor mode provide shared virtualization software. The virtualization mode indicates how the virtualized application is handled. It tells if the packaged files are extracted to a storage device or if they are extracted to memory. Isolation mode indicates how the virtualization solution handles the access to data in the operating system.

Table 8. *The features of the compared application virtualization solutions.*

Virtualization solution	Operating system	Execution mode	Virtualization modes	Isolation modes	Application streaming
BoxedApp	Windows	User	RAM	Varied	No
Cameyo	Windows Linux Mac OS	User	RAM / Disk	Data mode Isolated mode Full access mode	Yes
CDE	Linux	Supervisor	Disk	Seamless execution	Yes
Microsoft App-V	Windows	Supervisor	RAM	Isolated Strictly isolated	Yes
VMware ThinApp	Windows	User	RAM	Merged WriteCopy	Yes

All the application virtualization solutions apart from the CDE work on Windows operating systems. Only the Cameyo and the CDE work on the Linux and only the Cameyo works on the Mac OS. The Cameyo supports most operating systems as it has ability to stream applications through HTML5 [15]. The CDE and the Microsoft App-V require a kernel component to be installed to use them. Almost all of the virtualization solutions extract the runtime environment of the virtualized application to memory. The CDE is the only one which uses the environment directly from a storage device [40]. The Cameyo is the only solution which offers both options.

In the BoxedApp it is possible to select if the file system or the registers are virtualized. This allows isolating the applications from the actual file system or letting the applications to use it if needed. [11] The Cameyo provides three different levels of isolation. Data mode lets the applications to access and to write any data in the operating system's user folder or in network locations. Any other writes and reads are directed to an isolated folder. In isolated mode all the writes and reads are directed to the isolated folder.

Full access mode allows the applications to modify the files and registry entries on the host machine. [15] In the CDE all the writes and reads are directed by default to special isolated application folders. However, if the virtualized applications are launched from other some folder, they can use the files in that folder assuming that there are no virtual counterparts to them. This is called as seamless execution mode. [40] The Microsoft App-V has two modes. In isolated mode the virtualized applications can read any files from the file system. However, all the writes are done into isolated folders and not in the actual file system. In strictly isolated mode the applications can't read files from the file system and are restricted to only the files in the isolated folders. [6] The VMware ThinApp supports two isolation modes as well. The merged mode allows the virtualized applications to modify the filesystem and the registry entries of the host machine. In writecopy mode all the writes are directed to isolated folders. [120] All of the virtualization solutions apart from the BoxedApp support application streaming. As the virtualization solutions only focus on virtualizing the applications, they don't do any resource management. As such any real-time capabilities need to be provided by the operating system itself.

9.3.4 Virtualization Applicability to Industrial Automation

The previous research has shown that it is possible to meet the real-time requirements of many real-time applications when executed in a virtual environment. From the virtualization solution comparisons one can see that there is some effort at trying to implement virtualization solutions for industrial automation systems. Most notably the Wind River Open Virtualization and the XtratuM are designed for industrial automation systems which have hard real-time requirements. The Wind River's solution is more focused on mixed criticality systems where real-time operating systems and general purpose operating systems execute simultaneously on same hardware. The XtratuM is more focused on embedded real-time systems and as such the features it provides are minimal. This makes it easier to verify the trusted codebase it provides. Even the Xen provides some support for real-time with its real-time schedulers. With OS-level virtualization there are no solutions which would support hard real-time requirements natively and as such they are not suitable for applications with real-time requirements. The real-time capabilities of the application virtualization solutions are dependent of the operating systems they are used on. The more advanced virtualization features (save/restore, live migration and fault tolerance) tend to stop the execution of VMs for a moment when the data is copied from one location to another. As such they are not suitable for real-time applications. Some alternative methods need to be implemented to provide the fault tolerance and the availability.

Not all the applications in industrial automation systems have real-time requirements. These applications can utilize the more advanced features of the virtualization solutions. With these capabilities it is possible to provide the fault tolerance and the availability

for the industrial automation systems. The application virtualization is not necessarily useful in industrial automation systems to virtualize the main software as it provides the worst isolation of all the virtualization technologies. However, the application virtualization could be used to move software out from the factory floors. Instead of having piece of software installed in every human machine interface in the factory floor, the software could be located in centralized location and streamed to the human machine interfaces. This would allow replacing the human machine interfaces with simple monitors and input devices which are easy to replace if they become faulty. Almost all of the application virtualization solutions provide application streaming and the applications are extracted to memory instead of disk.

As there are virtualization solutions available for hard real-time applications and one virtualization solution is even aimed purely for embedded devices, the virtualization can be used for all the levels in industrial automation systems from the level 1 to the level 4. The largest problems with using virtualization in industrial automation are the lack of comprehensive studies of the performance and the real-time capabilities of different virtualization solutions and devices, the lack of industrial automation specific device and protocol support in the virtualization solutions, and the lack of virtualization support in the industrial automation devices. For example network virtualization is not useful in real-time applications if the devices don't provide schedulers suitable for real-time communications. Using virtualization in industrial automation systems require proper design, configuration and testing.

9.4 Implementing a Virtualized Industrial Automation System

Virtualization requires careful configuration and proper hardware for it to provide a virtualized environment which fulfills all the requirements of the system. As such virtualization should be taken into account at every step in the lifecycle model of an industrial automation system. Implementation of virtualized automation system is described in this chapter by using a lifecycle model. The requirements and considerations the virtualization brings are described for each step in the model. The lifecycle model used is shown in the Figure 30.

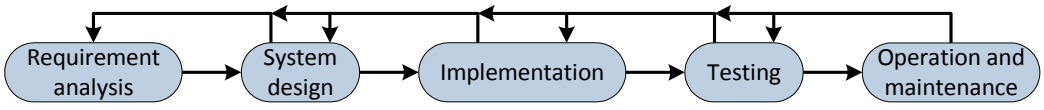

Figure 30. *The lifecycle model of an industrial automation system.*

The steps in the used lifecycle model are: requirement analysis, system design, implementation, testing, and operation and maintenance. From any step in the model it is possible to return to any previous step in the model if some problems are noticed or changes are needed. The whole process should be documented along with any security decisions made during the process.

9.4.1 Requirement Analysis

The first step is the requirement analysis. In this step all the requirements for the industrial automation system are identified. The requirements can be functional or non-functional. Some examples of requirements are the real-time requirements for the processes and availability requirements for important subsystems. Virtualization adds some requirements which should be taken into account when designing the virtualized industrial automation system. The Table 9 lists these requirements.

Table 9. *The virtualization specific requirements.*

No.	Requirement	Description
1.	Temporal isolation	Functional behavior of a VM or a container should not affect the performance of other VMs or containers.
2.	Spatial isolation	Software in a VM or a container should not be able to read or modify any data that belongs to other VMs or containers.
3.	Real-time	When real-time is required, the virtualization solutions should provide deterministic methods for executing the functional logic of the system.
4.	Network isolation	Virtual networks should not crosstalk. Management and monitoring networks should be isolated and secured from normal traffic. The physical resources allocated for virtual networks should be limited.
5.	Network access	Virtual networks should only be accessed through trusted virtual network endpoints.
6.	Management interface access	Access to the management interfaces of the virtualization solutions and host operating systems should be managed and access allowed only for people with proper training.
7.	Image/Snapshot access	Access to the VM or container images and snapshots should be managed physically and digitally. Any images or containers containing sensitive data should be protected with encryption.
8.	Image/Snapshot protection	The images and snapshots should be backed up in separate location. Images and snapshots should occasionally be scanned for viruses and cryptographic checksums should be calculated to identify any unwanted changes.
9.	Timekeeping and synchronization	Virtualization friendly timekeeping methods should be used in guest operating systems. The clocks should be synchronized in the guest and host machines.
10.	Fault tolerance and availability	At least three physical hosts should be used to provide fault tolerance and availability for important components in the system. One works as a primary copy, one as a secondary copy and one works as a backup machine when updating or maintaining components.
11.	Monitoring	The total resource usage of the host machines should be monitored per VM or container basis. Inter-host communications should be monitored.
12.	Virtual machine and container security	Virtualization doesn't remove the need for traditional security solutions between networks and machines. As such these should be implemented in the virtual machines and containers.

The requirement 1 is required for safety reasons. If some VM or container becomes infected and starts using more resources than it should use, its behavior should not affect the performance and the functionality of the other VMs and containers. The requirement 2 is needed for security as well as a virtualized environment can have VMs and containers with different security levels. If one of them becomes infected, the sensitive data could leak from other VMs or containers or the behavior of the software running in those could be modified. The requirement 3 is needed for the real-time applications as the virtualization solutions provide the base for the applications to run on.

The requirement 4 applies partially to traditional networks as well. Networks should be configured in a way that doesn't let data to be read from other networks if it's not intended to be read. The virtualization adds the extra requirement that the physical resources virtual networks can use should be limited. This is to make sure that all the virtual networks have enough physical resources to allow the networks to operate. As the network virtualization technologies don't have any inherent security features, the access to the virtual networks need to be managed. As such the requirement 5 is needed.

The configuration of the virtualized infrastructure is important. As such the access to the management interfaces should be controlled. Otherwise anyone could break the configuration and break the isolation between VMs and containers. Similarly configuring host operating systems could affect the performance of the VMs and as such the access to the host operating systems should be managed. The access should be given only for few people who have proper training for managing virtual environments. This creates the requirement 6.

The requirements 7 and 8 are focused on the images and snapshots made from VMs and containers. The images and snapshots can contain sensitive data like passwords. As such they need to be protected and the access to them allowed only for few trusted people. To validate that the images and the snapshots are safe, they should be scanned for viruses and malware periodically. Cryptographic checksums should also be calculated to notice any malicious modifications to them.

The requirement 9 comes from the fact that the timekeeping in virtualized environments can cause problems in some guest operating systems. Mostly the guest operating systems which use tick counting as timekeeping method. The tick counting can cause performance issues and the clock won't necessarily stay in proper time because of the virtualization. Any applications which require precise time should use operating systems which use timekeeping methods suitable for virtualized environments and the clocks should be synchronized in some way.

The virtualization lowers the amount of hardware required lowering the cost of building a system. The cost savings can be used to offer greater redundancy in the system by having multiple hosts running copies of the guest software. The three is recommended

amount for physical hosts to provide fault tolerance and availability for the software components. Two are required to provide the fault tolerance and failover incase the primary copy becomes faulty. The third machine can be used if one of the other two machines requires maintenance or becomes faulty. This allows having fault tolerance even when maintenance is carried out for one of the machines. This creates the requirement 10.

The monitoring the usage of resources inside VMs doesn't necessarily produce truthful values. As such the monitoring should be done in the virtualization software. The monitoring should also be done per container or VM basis to notice any abnormal behavior. The inter-host communication between VMs and containers should also be monitored in some way. As this communication differs from normal network communication between two machines, same monitoring tools cannot necessarily be used. These create the need for the requirement 11.

Lastly the requirement 12 is needed to remind that the virtualization is not a security solution on its own. The virtual machines and containers act like they are physical machines and as such they need to be treated in a similar manner.

9.4.2 System Design

In this step the architecture of the system is designed. The virtualization adds some changes how the industrial automation system architecture should be designed. The selection of virtualization technologies which are wanted to be used in the system direct how the system should be designed. The hardware virtualization solutions offer better isolation than the OS-level virtualization solutions. In OS-level virtualization solutions the applications should be divided into trusted applications and untrusted applications. Trusted applications are executed on their own physical machines separated from the untrusted applications and security measurements are added between the trusted and untrusted applications to prevent the trusted applications from becoming compromised. The reasoning for this is to allow using the better performance of the OS-level virtualization while avoiding the problems which malicious applications could cause with using all the resources on the host machine. If application virtualization is considered for its streaming capabilities, the architecture for the application storage and streaming should be designed.

The DMZ should be implemented into the system in similar manner to a non-virtualized environment. However, in a virtualized environment it is easier to improve the security. Physical firewalls are expensive and they increase the costs. Virtual firewalls can be duplicated easily. This allows implementing more firewalls into the systems with no increased cost. This enables creating multilevel DMZs easily. The most critical parts of the systems are behind multiple firewalls and multiple DMZs can be implemented between them and the office and the external networks. This can be taken as far as having

an own firewall for each application which can be configured only for the purpose of the application. Deciding which antivirus technology to use should also be part of the design. There are two choices: agentless or agent-based antivirus software. The agent-based antivirus software is installed on each VM or container. This takes resources from the actual computing in the VMs and the containers. Agentless antivirus software uses the VM introspection. This allows the antivirus software to reside outside the VMs and the containers taking less of their computing resources.

Monitoring should also be decided in this step. The inter-host communication monitoring relies largely on the technologies used. If the virtual switches provide monitoring features, these can be used for the monitoring. An alternative is to route the traffic through a physical switch with monitoring capabilities. The last alternative is to use special monitoring VM or container. Any traffic is routed through this VM or container and it has the monitoring tools installed in it. From this VM or container the traffic is then directed to the correct destination.

Real-time applications cannot use the more advanced features the virtualization solutions provide. Snapshotting, live migration and fault tolerance often halt the execution of a VM or a container for a moment. As such some alternatives should be used. One possible way of implementing these features could be to have two copies of the software running at the same time. The primary copy would execute normally without any interference. Any network messages directed to this software would be duplicated by using a SDN capable switch and the messages would be directed to the secondary copy. Any messages leaving from the secondary copy would be dropped by a SDN switch. Now the state of the secondary copy would follow the state changes of the primary copy. Snapshots could be taken from the secondary copy if required without it affecting the performance of the primary copy and simultaneously the secondary copy could be used as a failover if something happens to the primary copy. An example of updating secondary copy with SDN-capable switches is shown in the Figure 31. The arrows represent messages.

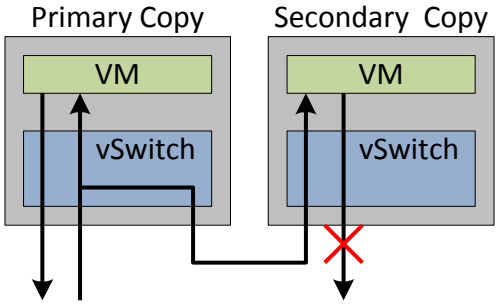

Figure 31. *Updating a secondary copy of a VM.*

The problem with this approach is the non-network-based events (mouse or keyboard inputs for example). These won't be delivered to the secondary copy. This method re-

quires support from the software itself or from the hypervisor if these kinds of events exist. Additionally a method needs to be implemented which monitors the state of the primary copy. The VMware ESXi supports this kind of fault tolerance by default. The default advanced features offered by the virtualization solutions can be used for non-real-time applications.

The virtual networks don't often have any security features and as such the access to these networks needs to be controlled. Special access points to the virtualized infrastructure must be designed into the architecture. For example a connection to Internet should go through a separate physical NIC to a VM or through a switch which adds virtual network labels to the packets and which implements security functions like authentication or a firewall. Similarly any outgoing data should go through these access points. Management and monitoring networks should also be separated and protected.

One should also consider how the storage space is virtualized. In hardware virtualization it is possible to store the data blocks that VMs write either in files or directly to storage devices. While the files can be easier to manage, the performance of accessing the data in the files is worse than if the data blocks are directly accessed from a storage device. The reason for this is the extra layer added by the file system in a hypervisor. One thing that needs to be considered as well is where the data is stored. If the data is stored locally for each host, accessing the data will be faster and the system is easier to build but more advanced features like live migration or fault tolerance require the data in storage devices to be copied from one place to another. If SAN is used, the data can be stored in a same place all the time. Data should be backed up in somewhere.

The last thing one should consider when designing the architecture is the image and snapshot management. In traditional industrial automation systems this is not required. The images and snapshots need to be stored and secured in some way. Also backups are needed to be created. The architecture should contain design choices how these features are implemented. Image and snapshot management policy should also be created to avoid creating excessive amounts of images and snapshots which increase the attack surface and which need to be secured.

There are many ways they architecture of a virtualized industrial automation system can be implemented. As an example the architecture of an existing virtualized industrial automation system is shown here. The virtualized industrial automation system is a test laboratory made in Tampere University of Technology. The laboratory has been built to test the virtualization in industrial automation systems and it had been designed before this thesis had started. The architecture of the system is shown in the Figure 32 on the next page. In the figure the lightning symbols represent wireless connections, vFW is a virtual firewall, the ICS network is the industrial control system network, and PCSs are process controllers. The virtualization technology used is hardware virtualization.

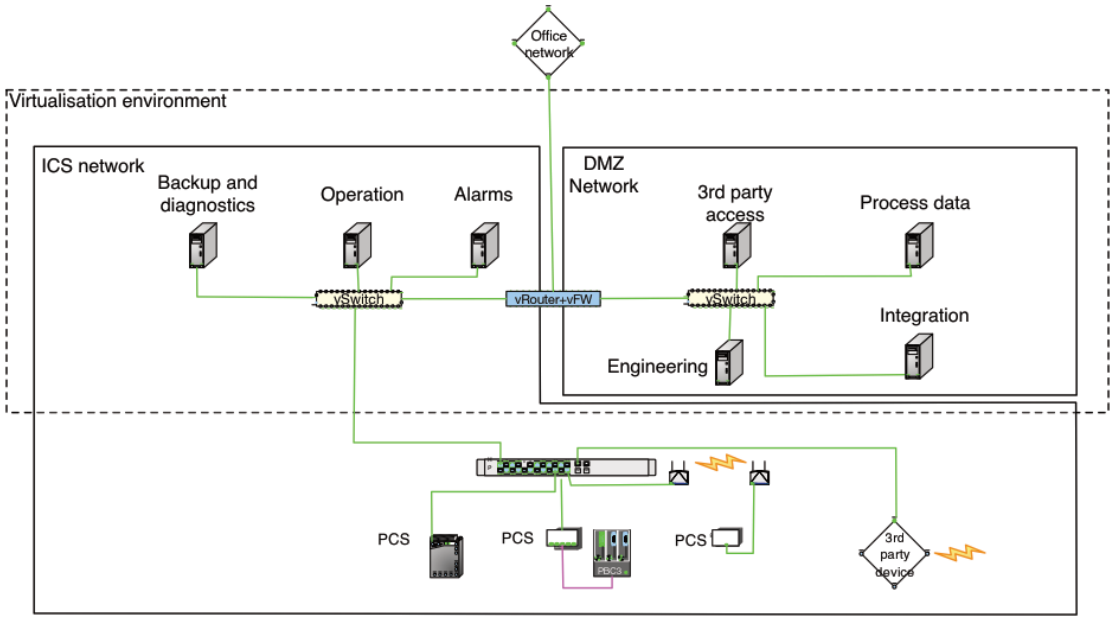

Figure 32. *Example architecture of a virtualized industrial automation system.*

The architecture shows only the layers 2-3.5 from industrial automation system architecture. The architecture implements a DMZ between the ICS network and the office network. All the machines in the DMZ are virtual machines and some of the machines that belong to the ICS network are virtual machines as well. There are three actual physical machines on which the virtual machines run on. The hypervisor used is the VMware's ESXi without any extra network virtualization solutions. As such the virtual switches are only basic layer 2 switches without any more advanced functions. Currently one of the challenges with the system is the functions that have real-time requirements. Because how the system has been implemented, it cannot provide any real-time guarantees for real-time applications.

9.4.3 Implementation

In this step the actual system is implemented. Hardware and software for the system is selected. To fulfil the temporal isolation requirements the virtualization solutions used should provide QoS functions for the physical resources used. Especially limiting the usage of the resources is important. The virtualization technologies should be selected based on the requirements of the system. From hardware virtualization the paravirtualization and hardware-assisted virtualization offer the best performance. However, the performance of hardware-assisted virtualization might vary depending on how often the execution needs to be transferred to the hypervisor. OS-level virtualization generally offers better performance than hardware virtualization. In hardware virtualization it is possible to provide better temporal isolation by using driver domains. These are the dedicated VMs which are responsible for running device drivers. The device domains restrict the effects of malfunctions and crashes between the drivers from affecting each

other. If the system should provide support for multiple different types of operating system kernels without any modifications, the only choices for virtualization technologies are the full and hardware-assisted virtualization methods in hardware virtualization. The used devices also affect which virtualization solutions can be used. The virtualization solutions should have virtualization support or device passthrough capability for the used devices. Alternatively the device manufacturer should provide paravirtual device drivers for the devices or the devices themselves should provide hardware-assisted virtualization. Spatial isolation is provided by the different virtualization technologies in different ways. The hardware-assisted virtualization provides it directly in the hardware while full virtualization, paravirtualization, OS-level virtualization, and application virtualization rely on checks in software. As such hardware-assisted virtualization should be preferred whenever possible. Type 2 hypervisors which are not implemented as kernel modules should be avoided for performance sensitive applications.

Proper memory configuration is important. The memory access in hardware-assisted virtualization can take longer because of the more complicated address translation. The effect of this can be mitigated by using larger memory pages. If the memory access is fully random, it might be better to use paravirtualization as it allows faster memory access with only a one level of address translation. The downside is that modifying the page tables is slower than in hardware-assisted virtualization. Advanced memory management should not be used if performance critical VMs or containers are hosted or real-time is required. Malicious software could use these methods to slow down the memory access for VMs or containers. IOMMU should be used to provide spatial isolation for the software components and the cache allocation technology should be used to provide temporal isolation when using the memory.

For real-time applications the virtualization solution should implement real-time scheduling algorithms. However, this is not the only way to provide the real-time capability for CPU usage. The other way is to simply dedicate a physical CPU to a real-time VM or a container. This way there won't be competition for the CPU usage and there is no need to schedule the CPU. With real-time scheduling algorithms one has to configure the scheduling parameters correctly for each VM and container to provide the real-time guarantees. Even if real-time schedulers are not used, proper selection of scheduling parameters is important as they can affect the performance of the VMs and the containers. Overcommitting the CPUs should also be avoided. There is a limit how many VMs or containers a CPU can handle before the CPU becomes fully utilized. When the CPU becomes fully utilized depends on the workloads of VMs and containers.

The used guest operating systems should support tickles timekeeping as it is better in virtualized environments than tick counting. Paravirtualization can be used for the tickles timekeeping. The hosts and the guests should synchronize the clocks. Guests can do this through the host or through a network. Proper settings need to be configured for the virtualization solution to use the selected method for the guests. Clock synchronization

service should be implemented into the system and it should be secured as retrieving time directly from the Internet is not suitable for industrial automation systems.

If the system has networks which span over untrusted networks, these should be secured with proper protocols like IPsec, STL, TTL, or SSH. The management and monitoring networks can be isolated from other networks by having physically isolated network connections or by using security protocols. The system should also be hardened and any unnecessary devices and features from the hosts and guests should be removed and disabled. This includes disabling promiscuous mode and MAC address changing in the NICs. Any tools and features provided by a virtualization solution that can be used to share data between host operating systems, VMs, and containers (For example clipboard on type 2 hypervisors.) should be disabled. Access control to all the virtualization solutions and any host operating systems should be set. In some cases the virtualization solution can offer tools for managing the access control but in some cases these have to be implemented separately.

When configuring I/O for the system, one should select simple schedulers in the guest operating systems. Schedulers which try to adapt the scheduling accordingly how the device is used and what is the state of the device can have worse performance in virtual environments than schedulers which use simple scheduling algorithms. If performance is required, device passthrough can be used. Device passthrough can also be used to allow using devices which don’t have paravirtual device drivers or for which there is no device emulation support in the virtualization solution. However, the device passthrough means that the device cannot be shared unless the device itself supports virtualization. This also disables many of the advanced features the virtualization solutions provide like live migration. IOMMU is required for the device passthrough.

9.4.4 Testing

One important aspect of implementing any industrial automation system is testing that the system meets all the requirements. Virtualized automation systems need to be tested as well. Testing the different aspects of isolation is important for virtualized industrial automation systems. To test the temporal isolation one should test three different aspects: CPU, memory and I/O. To test these one should have a program with some kind of workload. The workload should use resources for which the isolation is being tested. The time that it takes for the program to finish the workload should be tested in a case where the system is not stressed. The time to finish the workload can be measured by recording the time when the workload is started and the time when the workload is finished. The program is launched in one of the VMs or the containers on the tested machine. Next some testing tools are launched in some other VMs or containers which stress the CPU, the memory, or the I/O devices. These VMs or containers mimic the behavior of malfunctioning or malicious VMs or containers. The time to finish the workload is again measured and then compared to the value of the test where no stress

was applied to the system. Isolation Benchmark Suite is ready made tool for this kind of testing [55]. If it is noticed that the I/O requirements cannot be met, a way to improve the I/O performance is to dedicate CPU cores for the device drivers if it is possible. There are no easy methods for testing the spatial isolation. If the data is sensitive, one must make sure that only the VM that uses the data can access the data. If the data is not sensitive, it can be shared with other VMs or containers. Some kind of data protection is required in this case as well. This protection can be for example copy-on-write feature which allows the data to be shared safely between VMs and containers while preventing any modifications to the shared data. One should make sure that the shared data is not modified without any protection.

To test the real-time capabilities, one should have two physical machines. One is the tested machine. A message is sent to the tested machine from the other machine. The tested machine should have a control program that reacts to the message and answers to it by sending a new message to the other machine. The time between sending the first message and receiving the second message should be measured. As the address translations and other functionality can cause jitter, one should repeat the test many times to find the worst case scenario and to find the average round-trip time and jitter related to it. The isolation should also be tested simultaneously by running VMs or containers which stress the CPU, memory or I/O to be sure that the other VMs won't affect the real-time performance. An example of testing set up for testing the real-time capabilities is shown in the Figure 33.

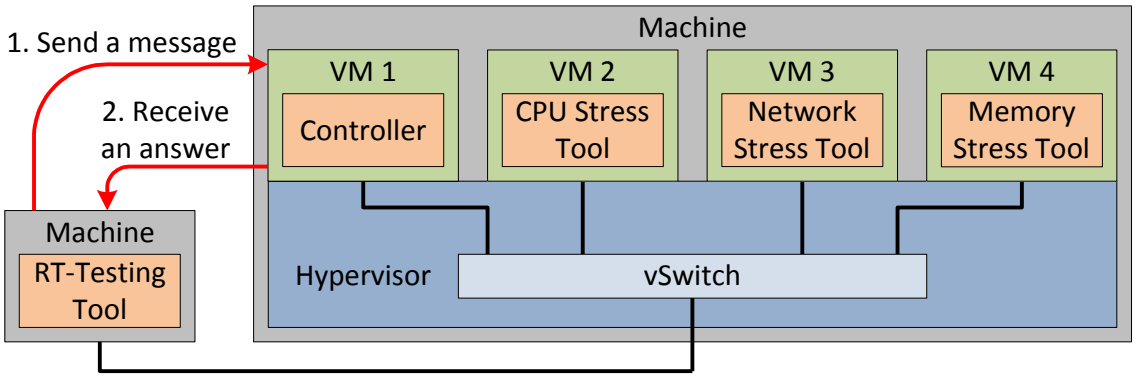

Figure 33. *An example set up to test real-time capabilities.*

In the figure there are two machines. One uses virtualization to host multiple virtual machines. The machine which doesn't use virtualization has a real-time testing tool installed on it. It sends a message to the VM 1 through the network connections (marked as black lines on the figure) and marks down the time when it sent the message. Then it waits for an answer message from the controller in the VM 1. The hypervisor is configured to provide real-time capabilities and the VM 1 uses real-time operating system. Once the machine, which doesn't use virtualization, receives the answer, it marks down the time when the message was received. Then the round trip time can be calculated from these times. The other VMs have different kinds of testing tools which stress the

different aspects of the machine. These tools are used to make sure that the behavior of the other VMs doesn't affect the behavior of the real-time VM.

Many of the already existing testing tools can be used to test security as VMs and containers behave like physical machines. One should test that the networks in the system don't leak data into other networks and monitor that the virtual networks have bandwidth limits. To do this one should stress the networks with a proper testing tool. One should also make sure that any NICs that can enable promiscuous mode or change MAC addresses are controlled only by trusted software components like hypervisors. Otherwise the NICs shouldn't be connected to trusted networks. Access to any management interfaces should be tested to make sure that they are secured. The timekeeping should be validated inside the VMs. One should let the systems run for a while and monitor the time inside the VMs and make sure that the times correspond to the actual time. The fault tolerance methods should also be tested to make sure that they work.

9.4.5 Operation and Maintenance

During this step the system is maintained. With virtualized environments it is important that the virtualization solutions are updated whenever they get updates. The virtualization solutions create the trusted code base for the system. If they have vulnerabilities, the whole system can be compromised. With SDN capability the updating can be done easily. There should be at least three machines which can be used to run VMs and containers. Two of the machines are actively used. One for the primary operation and one for the fault tolerance support. The third machine should work as a support in case one of the other two machines requires maintenance. When a virtualization solution needs to be updated, the updated version of the virtualization solution should be installed on the third machine. After this the copies of the VMs or the containers from the primary machine should be launched in the third machine. Any outgoing network connections from the third machine should be blocked at this point. Once the copies of the VMs and containers are running, one should check that everything works correctly. Lastly SDN can be used to forward to traffic to this third machine instead of the primary machine and any outgoing traffic from the primary machine can be blocked while any outgoing traffic on the third machine can be allowed. Now the primary machine can be shut down and updated. One possible problem with this method is the synchronization of the data. It also might be that the selected fault tolerance technology does not work correctly between different version of the virtualization solutions. This could mean that there is a period of time during which the fault tolerance is not working. An alternative would be using live migration to transfer either the primary copy or the secondary copy to the third machine assuming that the live migration works between different versions of the virtualization solution. Despite of these possible problems, these methods allow smaller down time than traditional systems as the system can be updated and configured during runtime. In addition images and snapshots can be used to speed up the process. In a tra-

ditional environment this kind of updating would not be feasible as each machine requires one extra machine which would increase the cost of building the system and make the system a lot more complicated. In virtualized environment the amount of used hardware is reduced and the configuration is easier to do with virtual networks so this kind of update method is more feasible.

Other important tasks at this step are monitoring and backup creation. One should monitor the resources the VMs and containers use. This can be used to identify any possible problems in the system. If new VMs or containers are created, one should be careful to not overcommit the resources of host machines and networks. Snapshots should be created for the VMs and the containers as well to back up any data changes there has been. Any data stored into storage devices should also be backed up to prevent loss of information in case of a hardware failure.

The users of the systems should also be trained. The users of the industrial automation system are typically plant technicians and process engineers. They don't typically know anything about virtualization as virtualization is not something that has been used in industrial automation systems. Some people should be trained to understand how the virtualization works, how to properly configure and how to monitor virtualized automation systems. These people should be responsible for managing the virtual environment.

9.5 Future Research Work

This thesis has created an overview of different virtualization technologies, compared some of the existing virtualization solutions, estimated their suitability for industrial automation, and identified some requirements and considerations one should take into account when designing a virtualized industrial automation system. There is still much that still needs to be researched. Currently there are no comprehensive studies of the available real-time-capable virtualization solutions. The real-time capabilities of these solutions should be thoroughly studied and tested and the real-time capabilities documented. In addition the OS-level virtualization solutions didn't seem to have hard real-time capabilities. No info was found of using rt-preempt patch and the Linux OS-level virtualization solutions together. It should be studied if these two technologies can be used together in similar manner to how KVM and the rt-preempt patch can be used.

One research subject could be to research if SDN could be used to create real-time capable network without requiring Industrial Ethernet protocols. The SDN uses centralized controller which makes it easier to determine the state of the network. Applications could inform the controller about their real-time requirements and the controller could set forwarding rules for the network according to this information. The controller could also calculate if it is possible to meet the real-time requirements in the current network by using some kind of algorithm. The SDN offers also more flexible forwarding policies than traditional networks. For example it is possible to forward the traffic com-

pletely differently based on what protocols are used to transfer the packets. As there is only one controller controlling the routing in the network, the network could be made deterministic more easily.

XtratuM is a hypervisor aimed for embedded devices and as such it has very little advanced features. It could be studied if the XtratuM could be used to provide security for field devices that can be connected to Internet with Industrial Ethernet. The architecture for this kind of device could contain two virtual machines, one running real-time operating system running real-time applications and one running general purpose operating system with security features. There could be separate network connections for the real-time operating system and the general purpose operating system. From within the industrial automation system any real-time data could be transferred through the real-time connection. Any non-real-time data would be transferred through the connection for the general purpose operating system. The general purpose operating systems could be used for example to monitor and get data from the device through the Internet. Access control, antivirus, intrusion detection and other security features could be implemented in it to provide security and these features wouldn't affect the operation of the real-time operating system. Additionally application streaming could be provided through the general purpose operating system to provide an interface for the device. It would basically implement a DMZ inside the device itself for outside connections while allowing real-time communications to be used with the rest of the industrial automation system. Research is required to find out if this kind of approach is possible or even feasible.

Some research could be done on how to implement a testing platform for industrial automation software by using virtualization. One possible way of using hardware virtualization for testing purposes could be to use two different VMs. One VM has the software that is being tested and one VM has a process simulator and testing tools running in it. The hypervisor would need to implement device emulation for all the common devices that are used in industrial automation systems and the backend for these emulated devices would direct the I/O calls to the testing tools and the process simulator in the other VM. To make the tests reliable, timing should be also properly controlled. The VM which executes the tested software should use virtual time specific for that VM. The hypervisor should control when the VM with the tested software executes and manage when the emulated devices return responses. This is to emulate the latencies in an actual physical system. To implement this kind of testing platform, some modifications are required for a hypervisor and device models for industrial automation devices are needed to be created. The hypervisor needs at least a method how the I/O calls from one VM are directed to a VM hosting the testing tools and the simulator. In addition the hypervisor needs an intelligent scheduler which manages when the VMs are executed and when the responses for the emulated devices are delivered to emulate the latencies of the physical systems. It could be possible to provide the testing as a service by selling VMs for customers to test their software on. The benefits of implementing the software

testing in this manner are the ability to control the execution of the VMs and the ability use emulated devices. This allows the software to be tested without any modifications to the software while still emulating the timing constraints of an actual physical system. The testing environment can also easily be modified for different scenarios because of the ability to create VMs and virtual connections between them.

The contents of this thesis are based on a literature review. As such nothing suggested in this thesis has been tested in actual systems. Future research work would be to see how well the suggestions presented in this thesis work in practice. For example the ideas presented for fault tolerance by using SDN needs to be tested and researched to see what kind of problems they have and if they are even feasible to implement. Similarly further research is required to see what kind of problems the updating method with three hosts has and if it possible to use it to minimize the down time caused by updates.

10. CONCLUSIONS

In this Master of Science Thesis most of the current virtualization technologies have been reviewed. As the virtualization technologies focus on emulating different parts of the computer, storage, and network architectures, one has to understand how the emulated hardware and software components work. Because of this, understanding virtualization technologies require deep understanding of the computer, storage and network architectures and technologies.

In virtualization solution comparison it was noticed that there are some solutions which are aimed for hard real-time applications. One of the solutions is even aimed to be used on embedded devices. Some studies have also showed that it is possible to reach under millisecond cycle times while using virtualization. These kinds of cycle times are suitable for most real-time applications in industrial automation. As such the virtualization could be used from level 1 to the level 4 in industrial automation systems. Many of the more advanced features like snapshotting and live migration cannot be used with real-time applications as they will cause pauses in virtual machine execution. One problem is also that current virtualization solutions don't necessarily have support for I/O devices used in industrial automation. To use I/O devices in virtual environment one has to have at least one of three things: device emulator, paravirtual device driver, or virtualization support in the device itself. As virtualization hasn't been used in industrial automation, industrial automation specific I/O devices don't have these things.

The virtualization provides security by isolating different pieces of software that execute on same hardware. This allows creating virtual machines, containers and virtual applications. However, the virtual machines and containers act like they are physical machines. This means that the security in them must be implemented in same way as in the actual physical machines. In industrial automation this includes implementing demilitarized zones to isolate untrusted and trusted networks. When designing a virtualized industrial automation system, the virtualization has to be taken into account in every step in the system's lifecycle. Virtualization adds some new requirements for the system which affect the different steps in the system's lifecycle. The virtualization affects everything from the architecture of the system to the maintenance of the system. Especially if virtualized components have any real-time requirements, the testing is important to make sure that the requirements can be met. The physical resources can also be configured to be shared in many different ways in virtualized environment and as such proper configuration is important. Proper hardware and virtualization solution selections are also important when virtualizing industrial automation systems.

REFERENCES

[1] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, J. Wiegert, Intel Virtualization Technology for Directed I/O, Intel Technology Journal, Vol. 10, No. 3, August 2006, pp. 179-192.

[2] Achieving Real-Time Performance on a Virtualized Industrial Control Platform, Intel, White paper, pp. 1-7. Available (accessed on 18.11.2015): http://www.intel.com/content/www/us/en/embedded/industrial/industrial-solutions-real-time-performance-white-paper.html

[3] K. Adams, O. Agesen, A Comparison of Software and Hardware Techniques for x86 Virtualization, ACM SIGOPS Operating Systems Review – Proceedings of the 2006 ASPLOS Conference, Vol. 40, No. 5, December 2006, pp.2-13.

[4] O. Agesen, A. Garthwaite, J. Sheldon, P. Subrahmanyam, The Evolution of an x86 Virtual Machine Monitor, ACM SIGOPS Operating Systems Review, Vol. 44, No. 4, December 2010, pp. 3-18.

[5] AMD I/O Virtualization Technology (IOMMU) Specification, AMD, PDF, February 2015, pp. 1-266. Available (accessed on 7.10.2015): http://support.amd.com/TechDocs/48882_IOMMU.pdf

[6] Application Virtualization, Microsoft, Website. Available (accessed on 27.11.2015): https://technet.microsoft.com/en-us/windows/hh826068

[7] Application Virtualization Technologies, CAMEYO, White paper, October 2013, pp. 1-6. Available (accessed on 16.11.2015): http://www.cameyo.com/resources/Application%20Virtualization%20Technologies.pdf

[8] ARM Infocenter, ARM, Website. Available (accessed on 1.10.2015): http://infocenter.arm.com/help/index.jsp

[9] P. Barham, B. Dragovic, K Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of Virtualization, ACM SIGOPS Operating Systems Review – SOSP ’03, Vol. 37, No. 5, December 2003, pp. 164-177.

[10] D. Boutcher, A. Chandra, Does Virtualization Make Disk Scheduling Passé?, ACM SIGOPS Operating Systems Review, Vol. 44, No. 1, January 2010, pp. 20-24.

[11] BoxedApp, Softanics, Website. Available (accessed on 27.11.2015): http://boxedapp.com/

[12] H. P. Breivold, K. Sandström, Virtualize for Test Environment in Industrial Automation, Emerging Technology and Factory Automation (ETFA), 2014 IEEE, Barcelona, September 2014, pp. 1-8.

[13] E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, E. Y. Wang, Bringing Virtualization to the x86 Architecture with the Original VMware Workstation, ACM Transactions on Computer Systems, Vol. 30, No. 4, November 2012, pp. 12:1 – 12:51.

[14] F. Bunn, N. Simpson, R. Peglar, G. Nagle, Storage Virtualization, Storage Networking Industry Association, Technical tutorial, August 2003, pp. 1-38. Available (accessed on 5.11.2015): http://www.snia.org/sites/default/files/sniavirt.pdf

[15] Cameyo, Komin LP, Website. Available (accessed on 27.11.2015): http://www.cameyo.com/

[16] A. Chandramouly, N. Patil, R. Ramamurthy, S. R. Krishnn, J. Story, Integrating Data Warehouses with Data Virtualization for BI Agility, Intel IT – Big Data and Businedd Intelligence, September 2013, pp. 1-8.

[17] Cisco IOS Quality of Service Solutions Configuration Guide, Release 12.2 - Policing and Shaping Overview, Cisco Systems, Inc., Website. Available (accessed on 16.10.2015): http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfpolsh.html

[18] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live Migration of Virtual Machines, NSDI'05: 2nd Symposium on Networked Systems Design & Implementation, pp. 273-286.

[19] J. Coleman, Reducing Interrupt Latency Through the Use of Message Signaled Interrupts, Intel, White paper, January 2009, pp. 1-23. Available (accessed on 6.10.2015): http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/msg-signaled-interrupts-paper.pdf

[20] Comparing the bandwidth and priority Commands of a QoS Service Policy, Cisco Systems, Inc., Website. Available (accessed on 16.10.2015): http://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-packet-marking/10100-priorityvsbw.html

[21] CoreOS, CoreOS, Inc., Website. Available (accessed on 10.11.2015): https://coreos.com/

[22] S. Crosby, D. Brown, The Virtualization Reality, Queue – Computer Architecture, Vol. 4, No. 10, December-January 2006-2007, pp. 34-41.

[23] W. Dai, L. Riliskis, V. Vyatkin, E. Osipov, J. Delsing, A Configurable Cloud-Based Testing Infrastructure for Interoperable Distributed Automation Systems, Industrial Electronics Society, IECON 2013 – 40th Annual Conference of the IEEE, October – November 2014, pp. 2492-2498.

[24] C. Dall, J. Nieh, KVM for ARM, Proceedings of the 12th Annual Linux Symposium, Ottawa, Canada, July 2010. Available (accessed on 28.9.2015): http://systems.cs.columbia.edu/files/wpid-ols2010-kvmarm.pdf

[25] B. Davie, A Stateless Transport Tunneling Protocol for Network Virtualization (STT), IETF Tools Pages, Website, March 2012. Available (accessed on 15.10.2015): https://tools.ietf.org/html/draft-davie-stt-01

[26] J.-D. Decotigne, The Many Faces of Industrial Ethernet, Industrial Electronics Magazine, IEEE, Vol. 3, No. 1, March 2009, pp. 8-19.

[27] Docker, Docker, Inc., Website. Available (accessed on 26.11.2015): https://www.docker.com/

[28] D. Domingo, L. Bailey, Red Hat Enterprise Linux 6: Performance Tuning Guide – Optimizing subsystem throughput in Red Hat Enterprise Linux 6, Red Hat, Inc., Website. Available (accessed on 26.11.2015): https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/index.html

[29] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, H. Guan, High Performance Network Virtualization with SR-IOV, Journal of Parallel and Distributed Computing, Vol. 72, No. 11, November 2012, pp. 1471-1480.

[30] M. Dontu, R. Sahita, Zero-Footprint Guest Memory Introspection from Xen, Xen Project DEVELOPER SUMMIT, Presentation, August 2014, pp. 1-39. Available (accessed on 16.11.2015): http://events.linuxfoundation.org/sites/events/files/slides/Zero-Foot-print%20Guest%20Memory%20Introspection%20from%20Xen%20_%20draft11.pdf

[31] M. S. Drescher, A flattened Hierarchical Scheduler for Real-Time Virtual Machines, Faculty of the Virginia Polytechnic Institute and State University, Thesis, 18 May 2015, pp. 1-77.

[32] B. Edgeworth, A. Foss, R. G. Rios, IP Routing on Cisco IOS, IOS XE, and IOS XR: How a Router Works, Cisco Systems, Inc., Website, January 2015. Available (accessed on 6.1.2015): http://www.ciscopress.com/articles/article.asp?p=2272154&seqNum=3

[33] Enabling VMware vShield Endpoint in a VMware Horizon View Environment: VMware vShield Endpoint 5.x and Horizon View 5.x, VMware, Inc., White paper, pp. 1-14. Available (accessed on 17.11.2015): http://www.vmware.com/files/pdf/techpaper/vmware-horizon-view-vshield-endpoint-antivirus.pdf

[34] S. Farinacci, T. Li, S. Hanks, D. Meyer, P. Traina, Generic Routing Encapsulation (GRE), IETF Tools Pages, Website, March 2000. Available (accessed on 15.10.2015): https://tools.ietf.org/html/rfc2784

[35] B. Galloway, G. P. Hancke, Introduction to Industrial Control Networks, Communications Surveys & tutorial, IEEE, Vol. 15, No. 2, July 2012, pp. 860-880.

[36] T. Garfinkel, M. Rosenblum, A Virtual Machine Introspection Based Architecture for Intrusion Detection, Network and Distributed System Security Symposium, February 2003, pp. 1-16. Available (accessed on 16.11.2015): https://labs.vmware.com/academic/publications/vmi-architecture

[37] O. Garg, Y. Wang, NVGRE: Network Virtualization Using Generic Routing Encapsulation, IETF Tools Pages, Website, September 2015. Available (accessed on 15.10.2015): https://tools.ietf.org/html/rfc7637

[38] S. Ghosh, P. Sampath, Evaluation of embedded virtualization on real-time Linux for industrial control system, Thirteenth Real-Time Linux Workshop, Prague, 2011, pp. 172-180.

[39] J. Gross, T. Sridhar, P. Garg, C. Wright, I. Ganga, P. Agarwal, K. Duda, D.Dutt, J. Hudson, Geneve: Generic Network Virtualization Encapsulation – draft-gross-geneve-02, IETF Tools Pages, Website, October 2014. Available (accessed on 16.10.2015): http://tools.ietf.org/html/draft-gross-geneve-02

[40] P. J. Guo, CDE: Run Any Linux Application On-Demand Without Installation, USENIX Large Installation System Administration Conference (LISA), PDF, December 2011, pp. 1-16. Available (accessed on 16.11.2015): http://www.pgbovine.net/publications/CDE-create-portable-Linux-packages_LISA-2011.pdf

[41] C. Gurr, Facilitating Microsoft Windows Vista Migration Through Application Virtualization, Dell Power Solutions, PDF, February 2008, pp. 88-91. Available

(accessed on 16.11.2015): http://www.dell.com/downloads/global/power/ps1q08-20080154-LANDesk.pdf

[42] N. Har'El, A. Gordon, A. Landau, M. Ben-Yehuda, A. Traeger, R. Ladelsky, Efficient and Scalable Paravirtual I/O System, USENIX Annual Technical Conference (USENIX ATC'13), 2013, pp. 131-142.

[43] S. F. Hasan, Emerging Trends in Communication Networks, Springer International Publishing, 2014, pp. 19-32.

[44] C. Helpa, State of the art hardware and virtualization extensions, Part of the Seventh Framework Programme Funded by the EC – DG INFSO, October 2012, pp. 1-52.

[45] C. Herber, D. Reinhardt, A. Richter, A. Herkersdorf, HW/SW Trade-offs in I/O Virtualization for Controller Area Network, Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE, June 2015, pp. 1-6.

[46] W. Huang, Introduction of AMD Advanced Virtual Interrupt Controller, AMD, XenSummit, Presentation, August 2012, pp. 1-26. Available (accessed om 7.10.2015): http://www.slideshare.net/xen_com_mgr/introduction-of-amd-virtual-interrupt-controller

[47] C. Hudson, P. Congdon, Edge Virtual Bridging with VEB and VEPA, Hewlett-Packard Development Company, PDF, May 2009, pp. 1-92. Available (accessed on 20.10.2015): http://www.ieee802.org/1/files/public/docs2009/new-hudson-vepa_seminar-20090514d.pdf

[48] J. Hwang, S. Suh, S. Heo, C. Park, J. Ryu, S. Park, C. Kim, Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones, Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE, 10-12 January 2008, pp. 257-261.

[49] IEEE Standard for Ethernet, The Institute of Electrical and Electronics Engineers, Inc., December 2012.

[50] IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks, The Institute of Electrical and Electronics Engineers, Inc., November 2014.

[51] Improving Real-Time Performance by Utilizing Cache Allocation Technology – Enhancing Performance via Allocation of the Processor's Cache, Intel, White paper, April 2015, pp. 1-16. Available (accessed on 3.11.2015): http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf

[52] Intel 64 and IA-32 Architectures Software Developer's Manual, Intel, PDF, Vol. 3, June 2015. Available (accessed on 30.9.2015): http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html

[53] Intel Virtualization Technology for Directed I/O: Architecture Specification, Intel, PDF, October 2014. Available (accessed on 7.10.2015): http://www.intel.com/content/www/us/en/embedded/technology/virtualization/vt-directed-io-spec.html

[54] Intelligent Queueing Technologies for Virtualization - An Intel-VMware Perspective: Enhanced Network Performance in Virtualized Servers, VMware, Inc., White paper, pp. 1-4. Available (accessed on 20.10.2015): https://www.vmware.com/files/pdf/partners/intel/vmdq-white-paper-wp.pdf

[55] Isolation Benchmark Suite, Clarkson Virtualization Center, Website. Available (accessed on 2.12.2015): http://web2.clarkson.edu/class/cs644/isolation/index.html

[56] N. Iwamatsu, Paravirtualized USB Support for Xen, Fujitsu Laboratories LTD., PDF, pp. 1-12. Available (accessed on 7.10.2015): http://www-archive.xenproject.org/files/xensummit_tokyo/26_NoboruIwamatsu-en.pdf

[57] R. Jain, S. Paul, Network Virtualization and Software Defined Networking for Cloud Computing: A Survey, IEEE Communications Magazine, Vol. 51, No. 11, November 2013, pp.24-32.

[58] Kernel Virtual Machine, Website. Available (accessed on 26.11.2015): http://www.linux-kvm.org/page/Main_Page

[59] N. Khang, APIC Virtualization Performance Testing and Iozone, Intel, Website, December 2013. Available (accessed on 7.10.2015): https://software.intel.com/en-us/blogs/2013/12/17/apic-virtualization-performance-testing-and-iozone

[60] S. T. King, G. W. Dunlap, P. M. Chen, Operating System Support for Virtual Machines, Proceedings of the 2003 USENIX Technical Conference, 2003, pp. 1-14.

[61] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, kvm: the Linux Virtual Machine Monitor, Proceedings of the Linux Symposium, Vol. 1, Ottawa, Ontario, Canada, June 27th-30th 2007, pp. 225-230.

[62] S. R. Kleiman, Vnodes: An Architecture for Multiple File System Types in Sun UNIX, USENIX Summer, Vol. 86, 1989, pp. 238-247.

[63] Knowledge Base – Understanding virtual machine snapshots in VMware ESXi and ESX (1015180), VMware, Inc., Website. Available (accessed on 29.10.2015): http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1015180

[64] T. Komperda, Virtualization Security, InfoSec Institute, Website, December 2012. Available (accessed on 16.11.2015): http://resources.infosecinstitute.com/virtualization-security-2/

[65] R. F. van der Lans, Data Virtualization in Business Intelligence Architectures: Revolutionizing Data Integration for Data Warehouses, Morgan Kaufmann, July 2012, pp. 1-25.

[66] H. Liu, H. Jin, X. Liao, Z. Pan, XenLR: Xen-based Logging for Deterministic Replay, 2008 Japan-China Joint Workshop on Frontier of Computer Science and Technology, 27-28 December 2008, pp. 149-154.

[67] Q. Lin, Z. Qi, J. Wu, Y. Dong, H. Guan, Optimizing virtual machines using hybrid virtualization, Journal of Systems and Software, Vol. 85, No. 11, November 2012, pp. 2593-2603.

[68] Linux Containers, Website. Available (accessed on 26.11.2015): https://linuxcontainers.org/

[69] LinuxCounter – Lines of code of the Linux Kernel Versions, The Linux Counter Project, Website. Available (accessed on 16.11.2015): https://www.linuxcounter.net/statistics/kernel

[70] Linux-VServer, Website. Available (accessed on 26.11.2015): http://linux-vserver.org/Welcome_to_Linux-VServer.org

[71] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Stridhar, M. Bursell, C. Wright, Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, IETF Tools Pages, Website, August 2014. Available (accessed on 15.10.2015): https://tools.ietf.org/html/rfc7348

[72] N. Mahmud, K. Sandström, A. Vulgarakis, Evaluating Industrial Applicability of Virtualization on a Distributed Multicore Platform, Emerging Technology and Factory Automation (ETFA), 2014 IEE, Barcelona, September 2014, pp. 1-8.

[73] R. McDougall, J. Anderson, Virtualization Performance: Perspectives and Challenges Ahead, ACM SIGOPS Operating Systems Review, Vol. 44, No. 4, December 2010, pp.40-56.

[74] A. McIntyre, LOGIIC Virtualization Project Public Report, The Automation Federation, PDF, February 2015, pp. 1-26. Available (accessed on 26.10.2015): https://logiic.automationfederation.org/public/Shared%20Documents/P8PublicReport.pdf

[75] R. Mijat, A. Nightingale, Virtualization is Coming to a Platform Near You: The ARM Architecture Virtualization Extensions and the importance of System MMU for virtualized solutions and beyond, ARM, White paper, 2011, pp. 1-12. Available (accessed on 23.9.2015): http://www.arm.com/files/pdf/System-MMU-Whitepaper-v8.0.pdf

[76] P. A. Morreale, J. M. Anderson, Software Defined Networking: Design and Deployment, CRC Press, 2014, pp. 1-172.

[77] J. Nakajima, A. K. Mallick, Hybrid-Virtualization—Enhanced Virtualization for Linux, Proceedings of the Linux Symposium, Vol. 2, Ottawa, Ontario, Canada, 27-30 June 2007, pp. 87-96.

[78] J. Nakajima, A. Mallick, X86-64 XenLinux: Architecture, Implementation, and Optimizations, Proceedings of the Linux Symposium, Vol. 2, Ottawa, Ontario, Canada, 19-22 July 2006, pp. 173-183.

[79] G. Natapov, KVM Weather Report, redhat, PDF, May 2013, pp. 1-58. Available (accessed on 7.10.2015): http://events.linuxfoundation.org/sites/events/files/cojp13_natapov.pdf

[80] G. Neiger, A. Santoni, F. Leung, D. Rodgers, R. Uhlig, Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization, Intel Technology Journal, Vol. 10, No. 3, August 2006, pp.167-177.

[81] M. Nelson, B.-H. Lim, G. Hutchings, Fast Transparent Migration for Virtual Machines, 2005 USENIX Annual Technical Conference, pp. 391-394.

[82] Network Functions Virtualisation, European Telecommunications Standards Institute, Website. Available (accessed on 20.10.2015): http://www.etsi.org/technologies-clusters/technologies/nfv

[83] L. Niemistö, Virtual Fieldbus – Applicability, Technologies and Evaluation: Master of Science Thesis, Tampere University of Technology, May 2013, pp. 1-58.

[84] P. Ondrejka, M. Prpič, R. Landmannm, D. Silas, Red Hat Enterprise Linux 6: Resource Management Guide, Red Hat, Inc., Website. Available (accessed on 26.11.2015): https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/index.html

[85] OpenVZ: Virtuozzo Containers, Website. Available (accessed on 26.11.2015): https://openvz.org/Main_Page

[86] PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology, Intel LAN Access Division, Intel, Technical paper, January 2011, pp. 1-28. Available (accessed on 6.10.2015): http://www.intel.com/content/dam/doc/application-note/pci-sig-sr-iov-primer-sr-iov-technology-paper.pdf

[87] G. J. Popek, R. P. Goldberg, Formal requirements for Virtualizable Third Generation Architectures, Communications of the ACM, Vol. 17, No. 7, July 1974, pp. 412-421.

[88] I. Pratt, K. Fraser, S. Hand, C. Limpach, A. Warfield, Xen 3.0 and the Art of Virtualization, Proceedings of the Linux Symposium, Vol. 2, Ottawa, Ontario, Canada, 20-23 July 2005, pp. 65-77.

[89] QoS Frequently Asked Questions, Cisco Systems, Inc., Website. Available (accessed on 16.10.2015): http://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-policing/22833-qos-faq.html

[90] K. Remde, VMware or Microsoft? – The Complete Series, Microsoft, Website, August 2013. Available (accessed on 26.11.2015): https://blogs.technet.microsoft.com/kevinremde/2013/08/12/vmware-or-microsoft-the-complete-series/

[91] E. Reshetova, J. Karhunen, T. Nyman, N. Asokan, Secure IT Systems: Security of OS-Level Virtualization Technologies, Springer International Publishing, 2014, pp. 77-93.

[92] S. Rixner, Network Virtualization: Breaking the Performance Barrier, Queue – Virtualization, Vol. 6 No. 1, January/February 2008, pp. 37-52.

[93] J. S. Robin, C. E. Irvine, Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor, Proceedings of the 9th USENIX Security Symposium, Denver, Colorado, USA, August 14-17 2000, pp. 1-17. Available (accesses on 28.9.2015): https://www.usenix.org/legacy/events/sec00/full_papers/robin/robin.pdf

[94] M. Rosenblum, T. Garfinkel, Virtual Machine Monitors: Current Technology and Future Trends, IEEE Computer, Vol. 38, No.5, May 2005, pp. 39-47.

[95] R. Russell, virtio: Towards a De-Facto Standard For Virtual I/O Devices, ACM SIGOPS Operating Systems Review – Research and developments in Linux kernel, Vol. 42, No. 5, July 2008, pp. 95-103.

[96] J. Rutkowska, Subverting Vista Kernel For Fun And Profit, Black Hat Briefings 2006, Presentation, Las Vegas, USA, August 2006, pp. 1-52. Available (accessed on 16.11.2015): http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf

[97] K. Scarfone, M. Souppaya, P. Hoffman, Guide to Security for Full Virtualization Technologies – Recommendations of the National Institute of Standards and Technology, National Institute of Standards and Technology, PDF, January 2011. Available (accessed on 29.10.2015): http://csrc.nist.gov/publications/nistpubs/800-125/SP800-125-final.pdf

[98] B. Scholten, The Road to Integration: A Guide to Applying the ISA-95 Standard in Manufacturing, ISA – Instrumentation, Systems, and Automation Society, March 2007, pp. 29-30.

[99] Security of the VMware vSphere Hypervisor, VMware, Inc., PDF, January 2014, pp. 1-26. Available (accessed on 9.11.2015): http://www.vmware.com/files/pdf/techpaper/vmw-wp-secrty-vsphr-hyprvsr-uslet-101.pdf

[100] J. Seppälä, M. Salmenperä, Towards Dependable Automation, Cyber Security: Analytics, Technology and Automation, 2015, pp. 229-249.

[101] I. Shields, Learn Linux, 101: Create partitions and filesystems, IBM developerWorks, PDF, December 2012, pp. 1-22. Available (accessed on 5.11.2015): http://www.ibm.com/developerworks/library/l-lpic1-v3-104-1/l-lpic1-v3-104-1-pdf.pdf

[102] I. Shields, Learn Linux, 101: Hard disk layout, IBM developerWorks, PDF, August 2015, pp. 1- 22. Available (accessed on 5.11.2015): https://www.ibm.com/developerworks/library/l-lpic1-102-1/l-lpic1-102-1-pdf.pdf

[103] J. E. Smith, R. Nair, Virtual Machines: Versatile Platforms for Systems and Processes, Morgan Kaufmann, San Francisco, USA, 2005, pp. 1-26.

[104] Software and Hardware Techniques for x86 Virtualization, VMware, Inc., Information Guide, pp. 1-9. Available (accessed on 2.10.2015): http://www.vmware.com/files/pdf/software_hardware_tech_x86_virt.pdf

[105] S. Soltesz, H. Pötzi, M. E. Fiuczynski, A. Bavier, L. Peterson, Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors, ACM SIGOPS Operating Systems Review – EuroSys'07 Conference Proceedings, Vol. 41, No. 3, New York, USA, June 2007, pp. 275-287.

[106] Y. Song, G. Alatorre, A. Singh, J. Olson, A. Corrao, Information Systems and Information Technology: Virtualization of Storage and Systems, Chapman and Hall / CRC, 2014, pp. 47-1 – 47-12.

[107] Standardizing Data Center Server-Network Edge Virtualization, IBM, White paper, October 2010, pp. 1-14. Available (accessed on 20.10.2015): http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=QCL12363USEN

[108] W. Stallings, Operating Systems: Internals and Design Principles, 7th edition, Pearson Learning Solutions, New Jersey, USA, February 2011, pp. 477-480.

[109] Storage I/O Control Technical Overview and Considerations for Deployment, VMware, Inc., Technical white paper, pp. 1-12. Available (accessed on 9.11.2015): http://www.vmware.com/files/pdf/techpaper/VMW-vSphere41-SIOC.pdf

[110] K. Stouffer, J. Falco, K. Scarfone, Guide to Industrial Control Systems (ICS) Security – Recommendations of the National Institute of Standards and Technology, National Institute of Standards and Technology, PDF, June 2011. Available (accessed on 18.11.2015): http://escaladeit.com/sites/default/files/SP800-82-final.pdf

[111] J. Sugerman, G. Venkitachalam, B.-H. Lim, Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, Proceedings of the 2001 USENIX Annual Technical Conference, PDF, Boston, Massachusetts, USA, 25-30 June 2001, pp. 1-15. Available (accessed on 28.10.2015): http://www.vmware.com/pdf/usenix_io_devices.pdf

[112] Support Resources: Hosting and Cloud Automation Platforms, Odin, Website. Available (accessed on 26.11.2015): http://www.odin.com/support/

[113] TechNet – Hyper-V Portal, Microsoft, Website. Available (accessed on 26.11.2015): http://social.technet.microsoft.com/wiki/contents/articles/120.hyper-v-portal.aspx

[114] Technical Overview of Virtual Device Contexts, Cisco Systems, Inc. White paper, pp. 1-18. Available (accessed on 20.10.2015): http://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-10-slot-switch/White_Paper_Tech_Overview_Virtual_Device_Contexts.pdf

[115] Timekeeping in VMware Virtual Machines: VMware vSphere 5.0, Workstation 8.0, Fusion 4.0, VMware, Inc., Information guide, pp. 1-32. Available (accessed on 9.11.2015): http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf

[116] Understanding Full Virtualization, Paravirtualization, and Hardware Assist, VMware, Inc., White paper, pp. 1-17. Available (accessed on 22.9.2015): http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf

[117] Understanding Memory Resource Management in VMware vSphere 5.0: Performance Study, VMware, Inc., Technical white paper, pp. 1-28. Available (accessed on 1.10.2015): http://www.vmware.com/files/pdf/mem_mgmt_perf_vsphere5.pdf

[118] P. Varanasi, G. Heiser, Hardware-Supported Virtualization on ARM, APSys '11 Proceedings of the Second Asia-Pacific Workshop on Systems, July 2011, Article No. 11.

[119] Virtualization for process automation systems, Rockwell Automation, Inc., PDF, January 2013, pp. 1-26. Available (accessed on 26.10.2015): http://literature.rockwellautomation.com/idc/groups/literature/documents/wp/proces-wp007_-en-p.pdf

[120] VMware, VMware, Inc., Website. Available (accessed on 26.11.2015): http://www.vmware.com/

[121] VMware NSX for vSphere (NSX-V) Network Virtualization Design Guide, VMware, Inc., PDF. Available (accessed on 7.1.2015): https://www.vmware.com/files/pdf/products/nsx/vmw-nsx-network-virtualization-design-guide.pdf

[122] VMware ThinApp Agentless Application Virtualization Overview, VMware, Inc., White paper, pp. 1-6. Available (accessed on 16.11.2015): http://www.vmware.com/files/pdf/application-virtualization-vmware-thinapp.pdf

[123] VMware vSphere VMFS: Technical Overview and Best Practices, VMware, Inc., Technical white paper, pp. 1-17. Available (accessed on 6.11.2015): http://www.vmware.com/files/pdf/vmfs-best-practices-wp.pdf

[124] C. Waldspurger, M. Rosenblum, I/O Virtualization, Communications of the ACM, Vol. 55, No. 1, January 2012, pp. 66-73.

[125] Wind River Linux: Technology Profiles, Wind River, Website. Available (accessed on 26.11.2015): http://www.windriver.com/products/linux/technology-profiles/

[126] R. Wojtczuk, J. Rutkowska, Following the White Rabbit: Software attacks against Intel VT-d technology, Invisible Things Lab, April 2011, pp. 1-27. Available (accessed on 16.11.2015): http://invisiblethingslab.com/resources/2011/Software%20Attacks%20on%20Intel%20VT-d.pdf

[127] WP/What are containers, OpenVZ: Vituozzo Containers – Wikipedia, Website. Available (accessed on 10.11.2015): https://openvz.org/WP/What_are_containers

[128] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, C. A. F. De Rose, Performance Evaluation of Container-base Virtualization for High Performance Computing Environments, 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Belfast, UK, 27 February 2013 – 1 March 2013, pp. 233-240.

[129] XenParavirtOps, Xen Project Wikipedia, Website. Available (accessed on 24.9.2015): http://wiki.xenproject.org/wiki/XenParavirtOps

[130] Xen Project wiki, Website. Available (accessed on 16.11.2015): http://wiki.xen.org/wiki/Main_Page

[131] S. Xi, Real-Time Virtualization and Cloud Computing, Washington University in St. Louis, All Theses and Dissertations (ETDs), paper 1366, September 2014, pp. 1-101.

[132] XtratuM – Hypervisor: The simpler, the better, FentISS, the Universitat Politècnica de València, Website. Available (accessed 26.11.2015): http://www.xtratum.org/

[133] M. Xu, V. Malyugin, J. Sheldon, G. Venkitachalam, B. Weissman, ReTrace: Collecting Execution Trace with Virtual Machine Deterministic Replay, VMware, Inc., PDF, June 2007. Available (accessed on 29.10.2015): https://labs.vmware.com/academic/publications/retrace

[134] Y. W. Yeong, L. Z. Qiang, Securing and Consolidating Industrial Automation Systems Based on Intel Architecture Using Open Source Technology, Intel, White paper. Available (accessed on 27.10.2015): http://www.intel.com/content/www/us/en/embedded/industrial/system-consolidation-series/secure-industrial-systems-intel-open-source-tech-paper.html

[135] R. Zurawski, M. Naedele, Integration Technologies for Industrial Automated Systems – IT Security for Automation Systems, CRC Press 2006, 2006, pp. 23-1 – 23-15.

APPENDIX A: VIRTUAL SWITCHING

Switches are devices which forward data based on the OSI-model's layer 2 protocol addresses. The basic Ethernet switch is typically virtualized in one of two ways. These methods are defined in the 802.1Q standard under a term called *Edge Virtual Bridging* (EVB) and they are called as *Virtual Edge Bridge* (VEB) and *Virtual Edge Port Aggregator* (VEPA). Both of these methods focus on implementing virtualized Ethernet switch at the machines which reside at the edge of a network. The goal is to provide switching functionality for virtual machines (VMs). [50] The VEB and VEPA can be implemented in many ways and the standard doesn't specify how one is supposed to implement them. It simply specifies what functions and properties the VEB and VEPA need to fulfil to be 801.Q compliant. In *Software Defined Networking* (SDN) switches are also used to forward data, but these switches are not restricted to forwarding the traffic based on layer 2 protocol addresses. The forwarding can use for example use information from layer 2, 3 or 4 protocol headers to forward the data. These switches can be implemented in similar manner to the VEB and VEPA, but the actual implementation details depend on the SDN technology being used.

The VEB is a method where the virtual switch is completely implemented at the edge machine [47]. The VEB can be implemented fully in software, fully in hardware, or partially in both. These implementations can be seen in the Figure 34 and they use media access control (MAC) –addresses and virtual local area network (VLAN) IDs to forward the traffic to correct locations [47].

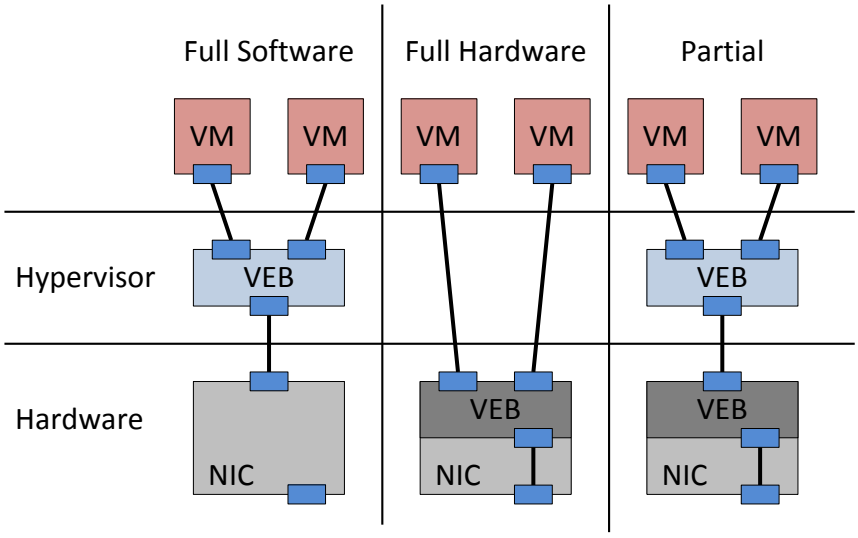

Figure 34. *The different implementations of VEB.*

The software implementation is the easiest method to implement, but it also has the worst performance as all the computations must be done in the central processing unit (CPU) which is also used for many other computations. When the VEB is implemented

fully in hardware, it is done by implementing it on *a Network Interface Card* (NIC) which uses *Single Root Input/Output Virtualization* (SR-IOV) technology [47]. The physical functions are used to configure the virtual switch inside the NIC and the VMs can use the virtual functions to access the NIC. The NIC can then forward the traffic to the external network or to a VM if it happens to reside inside the same machine. Partial implementation in hardware is possible with a technology called *Virtual Machine Devices queues* (VMDq) for example. The VMDq has been developed by Intel for NICs to move some computations from the software to the hardware. Most importantly it moves the forwarding decisions to the NIC and provides multiple receiving and transferring queues which can be dedicated for VMs [54]. The concept of VMDq can be seen in the Figure 35. The software component is still required to move the data between correct the queues and the VMs, and to do functions like VLAN tagging.

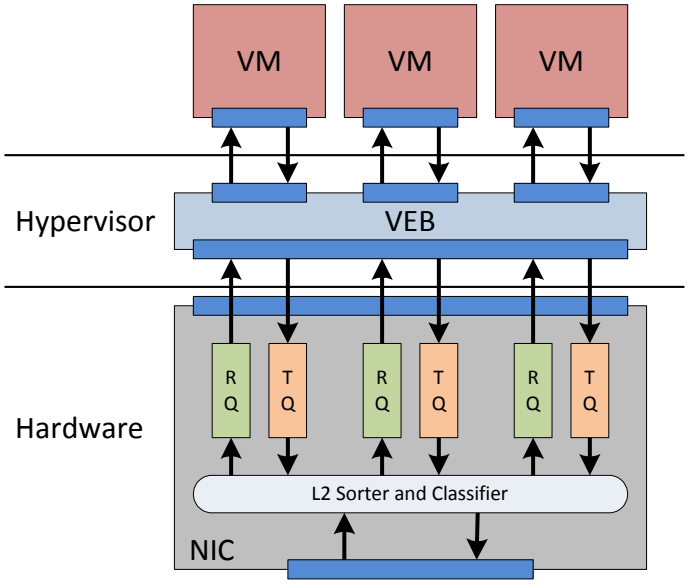

Figure 35. *Intel's VMDq technology.*

The problem with the VEB is that some data never leaves the edge machine. The effect of this is that any network tools and features developed for physical network cannot be used for this data. The support for these tools and features must be implemented separately for the VEBs and these are not always implemented for the virtual switches [107][47]. In software-based VEBs this is not as big of a problem as software is easy to create and modify, but the hardware-based VEBs can have problems. The hardware-based VEBs work in NICs which have very limited computing capabilities and they have many other responsibilities than just to work as a switch. This means that no complicated tools and features can be implemented in them. Another problem with the VEB is that the switch configuration is part of the edge machine configuration. This means that the network administrators cannot configure the virtual switches the same way as normal network devices are configured and the virtual switch configuration is part of the server administrator's job who manages the server [47]. The advantages of the VEB

are that it doesn't require an external switch and it allows fast traffic transferring between VMs which reside inside the same physical machine [107].

The VEPA takes a bit different approach. Instead of implementing the virtual switch completely in the edge machine, it implements it only partially in it. The VEPA uses the functionality of an existing switch to implement the virtual switch [107]. The comparison between VEB and VEPA can be seen in the Figure 36.

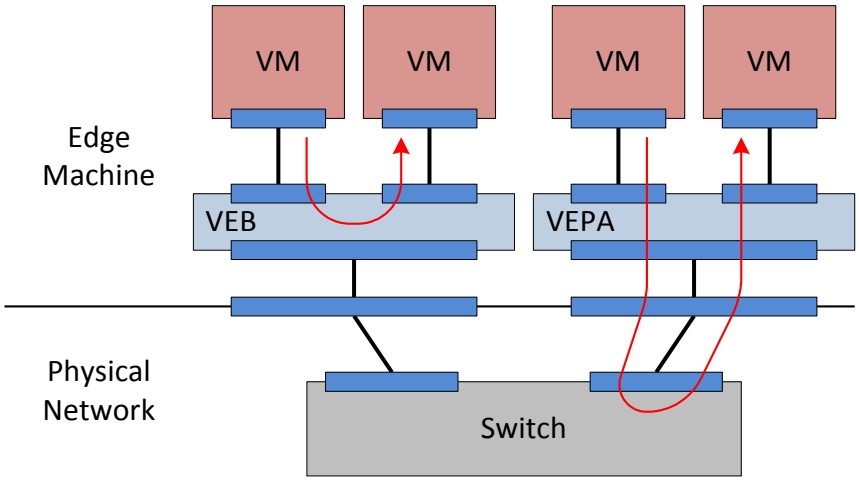

Figure 36. *Comparison between VEB and VEPA.*

Instead of forwarding the traffic inside the edge machine, the VEPA uses an existing switch to do the forwarding and to provide all the functionality of a switch. All the network traffic is first sent out from the edge machine to a switch and then in the switch it is forwarded back to the port it came from. Normally switches cannot send traffic back to the port the port from which they received it from. This requires the switch to have a small firmware update which updates the switch to have a reflective relay which allows it to send traffic back to the same port it received it from. [107] The traffic forwarding is done based on the MAC addresses and the VLAN IDs [47]. The edge machine requires a small component which forwards the received packet to the correct VMs and does functions like VLAN tagging. This can be done either fully in hardware or fully in software similarly to the VEB. [107]

The benefit of the VEPA is that it can use the features of the existing switches. This allows any tools and features developed for physical networks to be used with it [107][47]. The switch configuration is also moved out to the physical switches allowing network administrators to configure the switches the same way as they configure all the physical switches [107].

APPENDIX B: ADVANCED MEMORY MANAGEMENT

Hypervisors can provide many different advanced memory management techniques. These techniques allow overcommitting the memory on the physical host machines. This means that that the virtual machines can use more memory than is actually available on the physical machine. These advanced memory management techniques are: ballooning, page sharing, hypervisor swapping, and memory compression. One way how traditional operating system kernels allow memory overcommitting is by swapping memory pages [117], which haven't been used in a while, into a storage device. The memory pages are written to a file and read from it when the memory page is needed again. As the reading from the storage device is slower than reading from the memory, the memory access will be slower sometimes, but in exchange more memory can be used. The file on the storage device can be considered as an extension of the memory. Some of the advanced memory management techniques use memory swapping as well.

The ballooning is a technique where the hypervisor can control how much memory the guest operating system kernels inside the virtual machines (VMs) can use. In this technique the software running on top of the VMs is not aware of the host physical memory. The available guest physical memory is controlled by installing a special purpose driver called as *a balloon driver* in the guest operating system kernel [117][73]. The balloon driver can be used to *pin* guest memory pages inside the guest operating system kernel and it is controlled by the hypervisor. The communication between the hypervisor and the balloon driver is done by using paravirtualization methods. The pinning means that the memory pages are reserved for the balloon driver and they are always kept in guest physical memory and never swapped to a storage device. As the balloon driver doesn't actually do anything with these guest memory pages, the hypervisor can use the host memory pages corresponding to these guest memory pages in somewhere else. The balloon driver simply allows the hypervisor to control the host physical memory the VM uses between the minimum and the maximum memory values. [117] The concept of ballooning is show in the Figure 37.

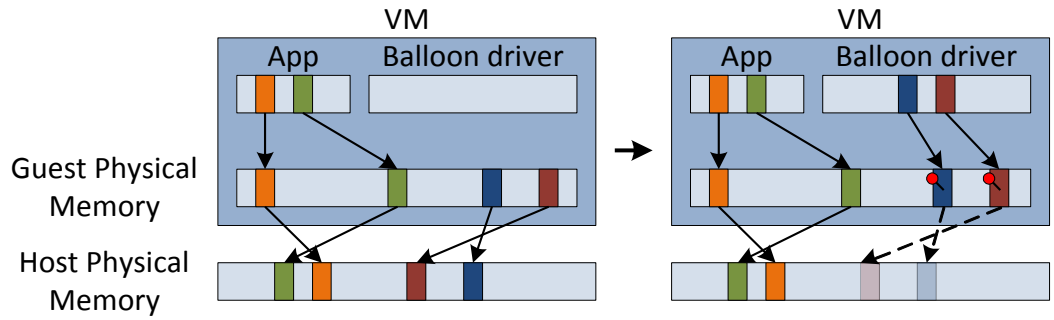

Figure 37. *The ballooning technique.*

If the guest operating system kernel has some free guest physical memory, this technique won't affect the performance in anyway. If the guest doesn't have free guest physical memory, it is under *memory pressure* and it has to swap memory pages to a storage device. This slows down the memory access of the software running inside the VM. [117] The benefit of this technique is that the guest operating system kernel can decide itself what memory pages to swap to the storage device [117][73]. It can do more intelligent decisions than the hypervisor as the hypervisor doesn't understand the contents of the VM's memory and can't know which memory pages are less likely to be used. [117]

The page sharing is a technique where a host physical memory page is shared with multiple different VMs. This is possible if the VMs' have a memory page which contains the same data. [117][73] This can be useful as many of the VMs might be running the same operating system kernel and much of the data might be duplicated in the memory. The page sharing simply allows saving some memory which can then be used elsewhere. This technique is transparent to the VMs meaning they don't know that the technique is being used. The host physical memory is scanned from time to time to identify duplicated memory pages and the scanning will take some resources to do. [117] If duplicates are found, the memory mappings are changed to one of the memory pages which will be shared between VMs and the rest of the memory pages are freed. When some VM tries to write to a memory page that is shared, copy-on-write feature [73] is used. The contents of the memory page are copied to an empty memory page and the VM's write is directed to this new memory page [117]. This will make the memory write slower that what it would be without the page sharing. [117] The page sharing is demonstrated in the Figure 38.

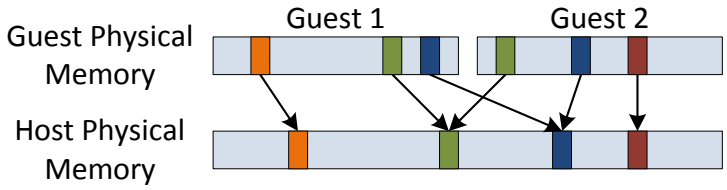

Figure 38. *The page sharing technique.*

The hypervisors can implement their own memory page swapping. This is called as *hypervisor swapping*. The hypervisor swapping works exactly the same as the swapping the operating system kernels do. The hypervisor has a file on the storage device where it writes memory pages when the host physical memory is full and new data is required in the memory. The hypervisor swapping is shown in the Figure 39. The usage of hypervisor swapping is not advisable as the hypervisor lacks the information of what memory pages are best candidates to be swapped to the storage device as it doesn't understand the contents of the VMs' memory. This can cause the hypervisor to swap memory pages to the storage device which the guest wants to access often causing huge performance issues for the guest. Another problem which might occur is that the hypervisor and the

operating system kernel both swap the same memory page to a storage device. This causes the memory page to be written twice in the storage device wasting some storage space. Additionally when swapping the memory page back to the memory, the swapping must be done twice. The benefit of this method is that it is immediate. The hypervisor can do the swapping when it is needed. The ballooning is reliant of the functionality of the guest operating system kernel which can delay the reclamation of the memory pages. The page sharing on the other hand relies for the VMs to have duplicated memory pages that can be shared and the page scans are done in predetermined times and not when they are needed. [117]

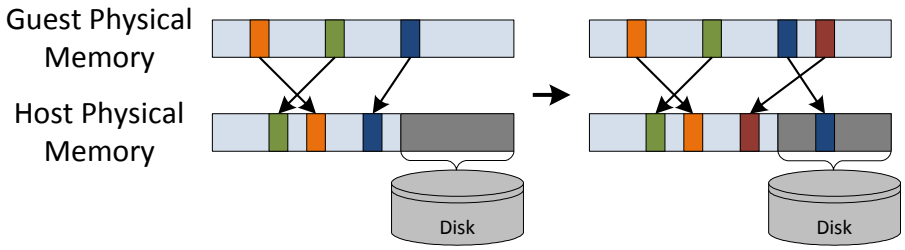

Figure 39. *The hypervisor swapping technique.*

The memory compression technique is very similar to the hypervisor swapping. The difference is that instead of swapping the memory page to a storage device, it is compressed and stored in the memory as shown in the Figure 40. [117]

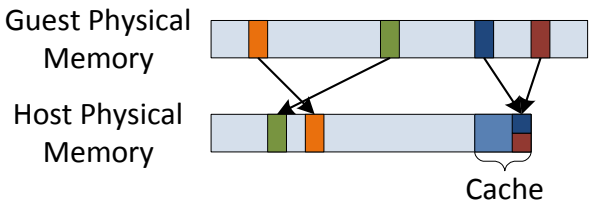

Figure 40. *The memory compression technique.*

Part of the host's physical memory is dedicated to work as a cache where the compressed memory is stored. When there is no more space in the memory for new memory pages, the least used memory page is compressed by using some algorithm and then moved to the cache. The compression allows the contents of a memory page to fit in much smaller space. When the compressed memory page is needed again, it is decompressed and moved out from the cache. [117] The memory compression is much faster than hypervisor swapping as reads and writes to a memory are much faster than reads and writes to a storage device [117][73]. The problem is that the degree of the compression is dependant of the contents of a memory page. This is why the compression is only done on memory pages where the compression is efficient and only when the hypervisor swapping would be used otherwise. If the cache in the memory becomes full, some old memory pages need to be decompressed and then swapped to a storage device to free some space in the cache. [117]